A Framework for Video-Driven Crowd Synthesis

by

Jordan J. Stadler

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Faculty of Graduate Studies (Computer Science)
University of Ontario Institute of Technology

Supervisor(s): Dr. Faisal Z. Qureshi

Abstract

A Framework for Video-Driven Crowd Synthesis

Jordan J. Stadler

Master of Science

Faculty of Graduate Studies

University of Ontario Institute of Technology

2014

We present a framework for video-driven crowd synthesis. The proposed framework employs motion analysis techniques to extract inter-frame motion vectors from the exemplar crowd videos. Motion vectors collected over the duration of the video are processed to compute global motion paths. These paths encode the dominant motions observed during the course of the video. These paths are then fed into a behavior-based crowd simulation framework, which is responsible for synthesizing crowd animations that respect the motion patterns observed in the video. Our system synthesizes 3D virtual crowds by animating virtual humans along the trajectories returned by the crowd simulation framework. We also propose a new metric for comparing the "visual similarity" between the synthesized crowd and exemplar crowd. We demonstrate the proposed approach on crowd videos collected under different settings and the initial results appear promising.

Acknowledgements

First and foremost I would like to take this opportunity to thank my advisor, Dr. Faisal Qureshi, for his guidance and encouragement. His continued support through undergraduate and graduate studies has been greatly appreciated.

Next I would like to thank my family and friends for always being supportive and understanding. My parents, Heidi and Randy, for always encouraging me to pursue my dreams. And lastly, to my three incredible sisters, who remind me daily to laugh and love.

Contents

1 Introduction **1**

1.1 Contributions 4

1.2 Overview 4

2 Related Works **6**

2.1 Crowd Analysis 6

2.1.1 Pedestrian Tracking 7

2.1.2 Behavioral Detection & Recognition 8

2.2 Crowd Synthesis 9

2.3 Synthesis via Analysis 11

2.3.1 2D 12

2.3.2 3D 12

2.4 Crowd Comparison and Evaluation 13

2.5 Afterword 14

3 Crowd Analysis **16**

3.1 Inter-frame Motion Extraction 16

3.1.1 Sparse Optical Flow 18

3.1.2 Dense Optical Flow 18

3.1.3 Scale-invariant feature transform 19

3.1.4 Motion Vectors: Observations 19

3.2 Motion Vector Clustering 21
3.2.1 Spectral Clustering 24
3.3 Path Generation 26
3.3.1 Motion Mask 29

4 Crowd Synthesis **31**
4.1 Agent Simulation 31
4.1.1 Goal Stack 32
4.1.2 Collision Avoidance 35
4.1.3 Reciprocal Velocity Obstacle Simulation 36
4.2 3D Human Animation 37
4.3 Path Diversification 40
4.4 From 2D to 3D and Back 47

5 Evaluation and Results **48**
5.1 Evaluation 48
5.2 Results 51
5.2.1 Campus Video 52
5.2.2 Grand Central Video 54
5.2.3 UCF Crowd Video 57
5.2.4 Histogram of Motion Score Discussion 62

6 Conclusions **63**
6.1 Future Work 63
6.1.1 Use Cases 64

7 Appendices **66**
7.1 Appendix A - Configuration File 66
7.2 Appendix B - Self-Tuning Spectral Clustering Algorithm 68

7.3 Appendix C - Globally Dominant Path Stitching Algorithm 69

Bibliography **70**

List of Tables

2.1 Crowd analysis for pedestrian tracking 9

2.2 Crowd analysis for behavioral recognition 10

2.3 Approaches to crowd synthesis 11

2.4 Crowd synthesis (via analysis) techniques. 13

3.1 Comparison of motion vector extraction on 60 seconds worth of video data. 21

5.1 Describes the three video files used for experimental results. 51

5.2 Scores for campus synthetic videos 52

5.3 Scores for the grand central synthetic videos 54

5.4 Scores for the UCF synthetic videos 60

List of Figures

1.1 Virtual crowd synthesized by analyzing an exemplar video. 1

1.2 Comparison of real and synthetic crowd motion. 2

1.3 An overview of our crowd synthesis via video analysis pipeline. 4

3.1 Motion vector extraction example 17

3.2 Sparse optical flow example 18

3.3 Dense optical flow example 19

3.4 SIFT motion vector example 20

3.5 SIFT performance in different scenarios 20

3.6 Resulting motion vector collections from motion extraction 22

3.7 Motion vectors overlayed on scene 23

3.8 Isolation of motion vector orientations 23

3.9 Result of discarded motion vectors from orientation clustering 25

3.10 Demonstration of locally dominant motion vectors 26

3.11 Stages of motion vector clustering 27

3.12 How neighboring cells are determined 28

3.13 Resulting globally dominant paths. 28

3.14 Motion mask generation 29

3.15 Motion masks and paths overlaid on scenes 30

4.1 Agent navigation with goal stack 33

4.2 Demonstrated the tension parameter on curves 34
4.3 Resulting paths of collision avoidance 35
4.4 Shows agents in single file 38
4.5 Demonstrates path diversification 38
4.6 The path visualization tool 39
4.7 Square method for diversification 40
4.8 Results of the square method 41
4.9 Triangle method for diversification 42
4.10 Results of the triangle method 43
4.11 Circle method for diversification 44
4.12 Results of the circle method of diversification 45
4.13 Path projection from 2D to 3D 46

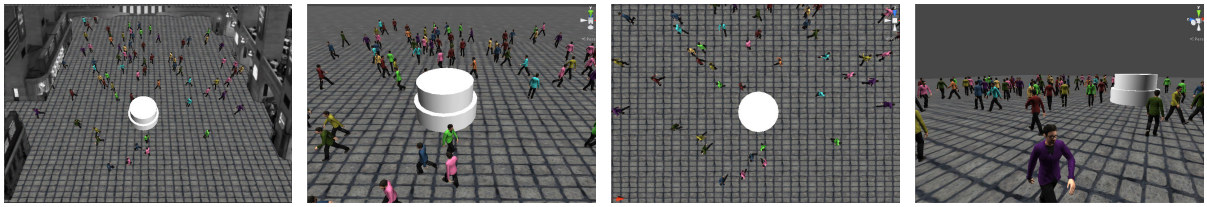
5.1 Histogram of motion and orientations 48
5.2 Demonstration of the sliding window for histogram of motion generation 49
5.3 Output of sliding window 49
5.4 Frames from campus video showing motion 53
5.5 Histograms of motion for campus video 54
5.6 Synthetic crowd motion for campus video 55
5.7 Frames from grand central video showing motion 56
5.8 Histograms of motion for the grand central video 57
5.9 Synthetic crowd motion for the grand central video 58
5.10 Frames from the UCF video showing motion 59
5.11 Histograms of motion for the UCF video 60
5.12 Synthetic crowd motion for the UCF video 61

Chapter 1

Introduction

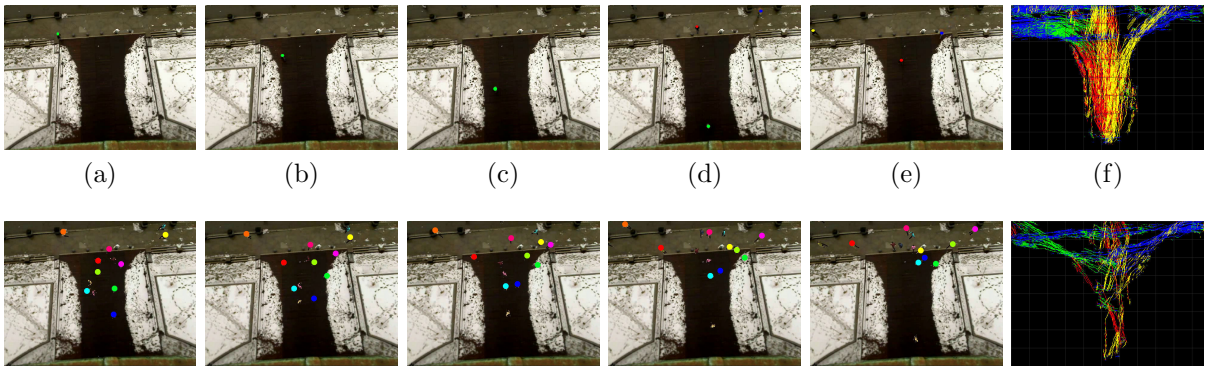
Many species, including humans, exhibit coordinated group behavior: schools of fish, flocks of birds, herds and packs of animals, and human crowds [47]. Some argue that such group behaviors are important for survival [3]. Humans also have a great facility for perceiving group behavior [50]. There is mounting evidence from the psychology literature that humans are able to perceive group behavior without decoding the individual motions. There is also some evidence that mechanisms that enable humans to perceive crowds are important and that any abnormalities in these mechanisms may adversely effect one's social functioning. There is much interest in the computer vision community to develop methods for analyzing crowd behavior.

Automatic crowd analysis and event detection is highly desirable, as it underpins a number of applications including crowd management, public space design, virtual envi-



(a) (b) (c) (d)

Figure 1.1: Virtual crowd synthesized by analyzing an exemplar video. (a)-(d) view the virtual crowd from different viewpoints.



(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k) (l)

Figure 1.2: Top row: (a)-(e) frames from the exemplar video and (f) extracted motion vectors. Bottom row: (a)-(e) stills from the synthesized crowd and (f) extracted motion vectors.

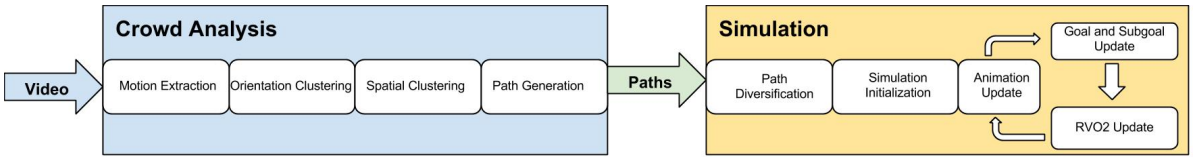
ronments, visual surveillance, and intelligent environments [57]. Crowd management, for example, can play a key role in mitigating crowd-related disasters, such as stampedes. It can be used to design safer public spaces. It is also possible to compute usage statistics and optimize space usage through crowd analysis. Additionally, crowd analysis can be combined with data-driven animation techniques to populate virtual environments. Crowd analysis and abnormal behavior detection, of course, is of great importance for video surveillance applications. Recent work on crowd analysis leverages crowd simulation techniques developed within the computer animation community to model the motion of individual entities in the crowd [16].

Raynold's seminal 1987 paper on *boids* showcased that group behaviors emerge due to the interaction of relatively simple, spatially local rules [41]. Since then there has been much work on crowd synthesis. The focus has been on methods for generating crowds exhibiting highly realistic and believable motions. Crowd synthesis, to a large extent, remains the purview of computer animators, who painstakingly fiddle with numerous parameters in order to achieve believable crowd motions. Many animation tools exist for generating high-quality crowds for computer entertainment industries. MASSIVE [30], Golaem Crowd [14], Miarmy [33], and Maya [31], for example, are popular crowd an-

imation tools. All of these tools have steep learning curves and these require a lot of manual tweaking to animate crowds with desired characteristics. Exemplar-based crowd synthesis appears a promising direction of future research [26]. Here, crowd synthesis parameters are learned by observing "real" crowds. These parameters can subsequently be used to synthesize crowds in previously unseen settings. Synthesized crowds have the added benefit of being unrestricted to a specific viewpoint. As shown in Figure 1.1 the same synthetic crowd can be observed from any angle.

Within this context, this thesis develops a framework for crowd synthesis via analysis. Videos exhibiting crowds are analyzed to extract high-level motion patterns. We employ vision-based motion analysis techniques to extract these patterns. These motion patterns are then combined with (spatially) local behavior rules, such as collision avoidance, path following, velocity matching, etc., to synthesize crowds. Specifically, we use Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation (RVO2) to synthesize crowd animations given the constraints extracted from exemplar videos [52]. RVO2 provides us with trajectories for individual agents and we use motion graphs to animate 3D virtual humans moving along these trajectories [20]. Figure 1.2 shows frames from videos of real and synthesized crowds. The proposed system extracted information from the video recording (top row) and used this information to synthesize virtual crowds (bottom row).

We also introduce a new metric for comparing the original crowd with a synthesized crowd. In order to compare the synthesized crowd with the original crowd, we render the synthesized crowd from a viewpoint similar to the one used to record the video of the real crowd. Motion parameters are extracted from the rendered footage and compared with those extracted from the real footage. Preliminary results seem to suggest that this metric is able to rank crowd pairs according to their motion similarities. More work is needed to further study this aspect of this work. The ability to compute the similarity between two crowds may be of some use in constructing a feedback loop that iteratively refines the synthesized crowds to better match real crowds.



(a)

Figure 1.3: An overview of our crowd synthesis via video analysis pipeline.

1.1 Contributions

This thesis develops, to the best of our knowledge, a-first-of-its-kind crowd synthesis via analysis pipeline (Figure 1.3). This pipeline combines motion analysis techniques developed in the machine vision community with behavior-based crowd animation methods that appear in the computer graphics community to animate crowds from exemplar crowd videos. Existing crowd cloning methods typically extract individual tracks from the exemplar crowd and use these tracks to animate virtual agents. These methods, therefore, cannot deal with large crowds where it is often infeasible to identify, less track, individual people. In contrast our method identifies dominant collective paths and uses these paths to synthesize virtual crowds. We also present a crowd similarity metric, which is different from entropy-based crowd similarity metrics developed within the crowd animation community [40]. Entropy-based crowd similarity metrics also assume that individual agents can be reliably tracked. Again, this assumption does not hold for large crowds. We have not compared our approach with other crowd animation techniques, as no existing scheme shares all of our assumptions. One way to compare our method with existing techniques, perhaps, is to perform human trials, where people are asked to rank the quality of the synthesized crowds. This, we feel, is beyond the scope of this thesis.

1.2 Overview

The rest of the thesis is organized as follows. We discuss related work in the next chapter. Chapter 3 and 4 describe crowd analysis and crowd synthesis, respectively. We present

results in the following chapter. Chapter 6 briefly discusses the strengths and limitations of this work. There we also identify directions for future work. We conclude this thesis with appendices containing relevant technical details.

Chapter 2

Related Works

This thesis develops a video-driven crowd animation technique. Our method uses crowd analysis to extract motion statistics from crowd videos and uses these statistics to synthesize virtual crowds using RVO2—a behavior-based crowd animation system. Below we briefly discuss existing work on crowd analysis and crowd synthesis.

2.1 Crowd Analysis

The goal of crowd analysis is to observe crowd data and extract useful (or relevant) information or metrics from it. Crowd data is typically videos or motion capture data of crowds. Many useful observations can be made through the use of crowd analysis. Crowd analysis generally falls into one of three categories: density estimation, pedestrian tracking, and behavior detection & recognition. Crowd density is a useful metric to determine how crowded an environment is. It can be used to determine how occupied an area is or how dangerous an emergency situation might be given a crowd size estimation. Furthermore it can be useful to simulation scenarios to estimate how many agents should be represented in a crowd reproduction. Density estimation techniques are not used in our system at this point. Pedestrian tracking is useful for observing how a space is being used and how crowds flow through environments. Behavioral understanding

such as crowd event detection can be used to assist with surveillance systems by alerting suspicious or dangerous events. This can make security teams more effective and response teams more responsive.

2.1.1 Pedestrian Tracking

Optical flow techniques estimate motion between two frames of a video. For videos of static environments, optical flow is an appropriate method for identifying objects moving within the scene. In scenes where the environment is moving, optical flow identifies how the scene is changing over time. Optical flow produces a grid (dense optical flow) or collection (sparse optical flow) of motion flow vectors. These vectors indicate how the scene has changed relative to a previous frame. Optical flow has various applications in crowd analysis, but here we focus on optical flow based pedestrian tracking [7, 34, 42, 16, 36]. The work of Eibl et al. [7] and Hu et al. [16] demonstrate various methods of clustering optical flow vector frames to generate dominant motion flow fields. These flow fields represent motion trends observed in a scene. Similarly Rodriguez et al. generate tracks of pedestrian motion through the observations of optical flow fields and a precomputed model of spatial crowd behaviors [42]. Moore et al. [34] show how optical flow can be used to detect anomalies in crowd motion by working under the assumption that crowds follow rules of hydrodynamics.

Many approaches to pedestrian tracking suffer from issues related to occlusions. Eshel et al. [8] propose a method of minimizing these problems through multiple overhead cameras. Head tracking is performed across multiple videos to produce the pedestrian tracks through the scene. Although the results are promising, this limits the crowd analysis to crowds that have been observed by these multi-camera setups. Our approach strives to work with single videos of crowds that are as unconstrained as possible.

Hu et al. [16] propose a method for determining motion patterns from videos of crowds in unstructured environments (i.e., a pedestrian has an equal chance to go in

any direction). Their method collects sparse optical flow vectors, combines these into a global motion flow field through clustering. Again the global flow field encodes the dominant motion patterns observed in a scene. These flow fields, for example, can be used during behavior understanding. Ozturk et al. [36] propose a similar technique for computing global flow fields from videos. However, instead of optical flow, they track Scale Invariant Feature Transforms (SIFT) [28] over multiple frames to generate motion vectors that are clustered to generate the flow field. Our method on crowd analysis is inspired by the techniques developed in [16, 36]. We have used both optical flow vectors and SIFT tracks to estimate motion flow fields; however, we found that SIFT tracks cannot be reliably extracted from synthetic videos. The subtleties of real crowd videos generate stronger SIFT keypoint description where as synthetic videos do not provide strong enough descriptors. The keypoint descriptors generated from synthetic videos are too similar to one another. The synthetic agents all have the same model and the floor texture is repeated throughout the scene, this appears to weaken SIFT keypoint matching. Further work into diversifying agents and textures could remove this issue. Consequently, we use optical flow to compare rendered footage of virtual crowds to the exemplar video.

2.1.2 Behavioral Detection & Recognition

There are many approaches to behavior detection [1, 32, 44, 21, 54] that strive to observe how crowds are behaving at a group or individual level. Andersson et al. [1] accomplish behavior detection through the use of Multiple Target Tracking (MTT), K-means clustering, and Hidden Markov Models (HMM). Saxena [44] and Dee [6] use Kanada-Lucas-Tomasi (KLT) tracker [29] for the purposes of behavior detection. These feature-based approaches are in contrast to the Social Force([32]), optical flow ([21]), and Energy Model ([54]) approaches to behavior detection and recognition.

Citation	Input	Approach	Result
[7]	Crowd video	Clustering optical flow vectors (various methods)	Generates dominant motion fields
[16]	Crowd video	Clustering optical flow vectors	Generates dominant motion fields
[34]	Crowd video	Treat crowd as fluid	Anomaly detection
[43]	Crowd video	Data-driven model with learned crowd behaviors	Temporal flows of motion through the scene
[8]	Crowd Videos (multi-camera)	Head tracking from many angles	Tracks of pedestrian motion through the scene.
[42]	Crowd video	Generate model information from video dataset and track via model predictions	Tracks of motion through the scene. Shown to work on crowds and even cell populations.
[36]	Crowd video	Observe SIFT motion vectors	Globally dominant motion paths. Shown to work on crowds and even cell populations.

Table 2.1: Crowd analysis for pedestrian tracking

Some approaches are successful at recognizing specific behaviors [54, 6, 2, 2]. Here we define recognition to be a more specific case of detection. When recognition is used, something specific is being detected. For example, detection would determine that an abnormal behavior has occurred while recognition would identify gathering or running within a crowd as that abnormal event. Xiong et al. [54] are capable of recognizing Running and Gathering while Dee [6] et al. are capable of recognizing Running, Loitering, Dispersal (inward), Dispersal (outward), and Formation. Andrade et al. [2] manage to recognize a block and unblocked exit through testing their methods on simulated scenarios while Liao [27] et al. successfully recognized fights in crowds.

2.2 Crowd Synthesis

Virtual crowds have been used for years in movies, video games, security simulations, etc. Automating pedestrians [45] is becoming increasingly popular. Recent developments in crowd simulations have led to more realistic looking and scalable crowds. Many plugins—

Citation	Input	Approach	Result
[1]	crowd video	MTT, K-means, HMM	behavior detection in dense crowds
[44]	crowd video	multi-frame feature point detection and tracking based on KLT tracker	crowd behavior recognition
[32]	crowd video	using a social force model to estimate interaction forces of particles and obtain per pixel Force Flow	abnormal behavior detection
[21]	crowd video	histograms of motion representing scene optical flow	congestion detection
[54]	crowd video	calculate a crowd density estimation, crowd distribution index, and kinetic energy of the crowd	Gathering or Running recognition
[6]	Crowd video	KLT tracklets and Histogram of Motion Directions	recognize Running, Loitering, Dispersal (outward), Dispersal(inward), and Formation
[2]	simulated crowd	observe crowd optical flow and use unsupervised feature extraction to encode behavior	detection of blocked exit
[27]	crowded scene	four MPEG-7 descriptors and SVM prediction	fight (brawl) recognition
[13]	crowd feed	Histogram of Oriented Gradients (HOG) based tracker	Crowd event recognition

Table 2.2: Crowd analysis for behavioral recognition

such as, Goalaem crowd [14] for the popular animation suite Autodesk Maya—exist to ease crowd animations from the animators perspective. Current approaches, however, still require a lot of tuning on the part of the animator.

Motion tile/patch based crowd synthesis approaches [55, 46, 19, 24] have received popularity in recent years for their scalability. These approaches to crowd synthesis are great for large-scale dynamic crowds however they are not ideal for synthetic crowd reproduction. The focus of these approaches is tight spatial and temporal existence between agents. For example Kim et al. [19] produce dense crowds with their deformable motion patches which stitch tiles together by patch entrances and exits. A given patch contains an action which can have its duration and location manipulated to some degree. The result is a crowd that is very interactive, containing a lot of activity. Although these

crowds are interesting to the viewer, they are not well suited for reproducing a crowd in that their focus is simply optimizing the actions that are occurring for and among agents.

Velocity fields [37, 53, 5] offer quality agent navigation results with the benefit of offering navigation from any position in the environment. The work of Patil et al. [37] offers a solution to directing crowd simulations using velocity fields. Similarly Wang et al. [53] perform direct crowd simulation with velocity fields generated from videos. Chenney [5] et al. propose a flow tile-based approach to crowd synthesis for representing and designing velocity fields. This approach is interesting in that it has the scalability associated with tile-based approaches while offering the quality of velocity fields. Steering based approaches [39, 35] are offline techniques and it is not immediately obvious how to use these techniques to create interactive crowds.

Citation	Approach	Result
[55]	construct motion patches	densely populate large environments
[46]	precomputed interaction between characters	large number of characters closely interacting
[19]	precomputation of deformable motion patches	dense crowd of characters interacting
[24]	construction of motion patches as building blocks for simulation	generation of complex virtual environments
[39]	steering model based on linear velocity prediction	real and virtual mixed reality simulation
[35]	optical flow based agent steering	vision based approach to collision avoidance

Table 2.3: Approaches to crowd synthesis

2.3 Synthesis via Analysis

Synthesis via analysis is a term used here to represent solutions that perform crowd synthesis specifically using crowd analysis. Synthesized crowds using the synthesis via analysis approach provide either 2D or 3D results. 2D approaches provide very constrained results although still interesting. We found 3D approaches to be more useful to our application as it offers more room for observing and manipulating the resulting

synthesized crowd.

2.3.1 2D

Flagg et al. [11] propose a video-based crowd synthesis technique utilizing crowd analysis for the purpose of generating crowd videos with realistic behavior and appearance. Their approach generates pedestrian sprites by segmenting pedestrians from input video. These resulting sprites are used in the output video as the agents. Furthermore, crowd tubes are used to avoid collisions and ensure accurate agent navigation. Their approach produces promising results but it is constrained to the field of view of the original video. 3D simulation based approaches to crowd synthesis offer the advantage of flexibility. Butenuth et al. [4] also produce a simulation restricted to 2D. They perform a hybrid solution to crowd synthesis although their output is a 2D simulation with discs for agents. Their focus is on large, dense crowds captured from a distance.

2.3.2 3D

3D approaches [26, 22, 25, 48] to synthesis via analysis offer flexibility in that they can be heavily manipulated and customized by the end-user. This is largely beneficial in that a user can reproduce a crowd and evaluate the reproduction before making adjustments (such as a very different camera angle) and altering the use of the synthesized crowd. However, these approaches rely on very constrained input video. The work of Lee et al. [22] relies on a top-down facing camera to observe the crowd and extract trajectories. Lerner et al. [25] make use of this technique but also rely on user input to annotate extracted trajectories for the purpose of agent behavior detection and recognition. Similarly having a reliance on motion capture ([26], [48]) data to feed a simulation can be restrictive as input. Motion capture based solutions offer high accuracy, but at the cost of requiring a highly structured video. Ideally a system would be capable of accepting an unconstrained video of a crowd and being able to reproduce it, which is the focus of

our work.

Citation	Dimensions	Agent Representation	Input
[11]	2D	sprite	raw crowd
[4]	2D	disc	high aerial view
[26]	3D	3D model	motion capture
[22]	3D	3D model	high positioned down facing camera video
[25]	3D	3D model	high positioned down facing camera video and behavioral annotations
[48]	3D	3D model	motion capture

Table 2.4: Crowd synthesis (via analysis) techniques.

2.4 Crowd Comparison and Evaluation

Methods for crowd entropy, a measure of a crowd's disorder, can be useful as a metric for observing and identifying crowd activities. Various methods for calculating crowd entropy have been proposed and used for different purposes. Guy et al. [15] propose a method for computing an entropy score for a given crowd navigating through a scene. Their method is used to evaluate steering methods and requires real-world data. The real-world data however is not a raw video to be processed but rather the result of a crowd being processed. Their method relies on precise comparisons of where an agent is versus where an agent should be. This method seems to work well for steering behaviors as the motions of an individual agent can be properly compared. Our system, however, generates agents navigating the scene along paths which are generated from a raw crowd video. As such we are more interested in comparing the output video of our system versus the input crowd video. Ihaddadene et al. [17] perform real-time crowd motion analysis in which they maintain an entropy value to watch for specific variations. Their method is not used for the evaluation of crowds but by observing the entropy they can estimate sudden changes and abnormalities in the crowd's motion. Ren et al. [40] propose a similar solution to crowd behavior detection through observing crowd behavior entropy. Both of these approaches are limited in that they rely on previous temporal information from

the scene. Our approach needs to directly compare one crowd to another as opposed to observing one crowd relative to itself.

An interesting proposal was put forth by Pelechano et al. [38] to validate crowd simulation through an immersive user study. Participants are placed in the virtual crowd using head mounted displays or similar devices. They interact with the crowd to establish presence and hope to determine crowd validation methods through similar study. This technique, although interesting, is not a feasible method of evaluation due to its equipment needs and the immaturity of the technique.

It is not uncommon for crowd simulations to be evaluated with a visual comparison performed by study groups. Lee at al. [23] utilize visual comparisons to evaluate their data-driven crowd simulations. This approach is good for determining which video matches a raw crowd better given a collection of outputs video; however, it would not perform as well with an iterative approach that strives to automate the process of best reproducing the input crowd. Performing group visual comparisons is lengthy and does not leave automation as a possibility. Similarly Karamouzas et al. [18] perform a visual comparison as a method of simulation evaluation.

2.5 Afterword

It is clear that both crowd analysis and synthesis are active areas of research within machine vision and computer graphics communities, respectively. Our work on crowd analysis is inspired by the work by Hu et al. [16] and Ozturk et al. [36]. We use the Lucas-Kanade [29] sparse optical flow technique to gather motion vectors from the exemplar video and use these motion vectors to compute global flow field. The global flow field is subsequently used during crowd synthesis. In contrast to existing techniques on video-driven crowd animation, our method does not assume that individuals can be reliably tracked within the crowd. Rather, we are solely interested in extracting overall motion

pattern from the scene. A second noteworthy feature of our method is that it generates 3D crowd animations [26, 22, 25, 48]. Lastly, we propose a new metric for comparing two crowds.

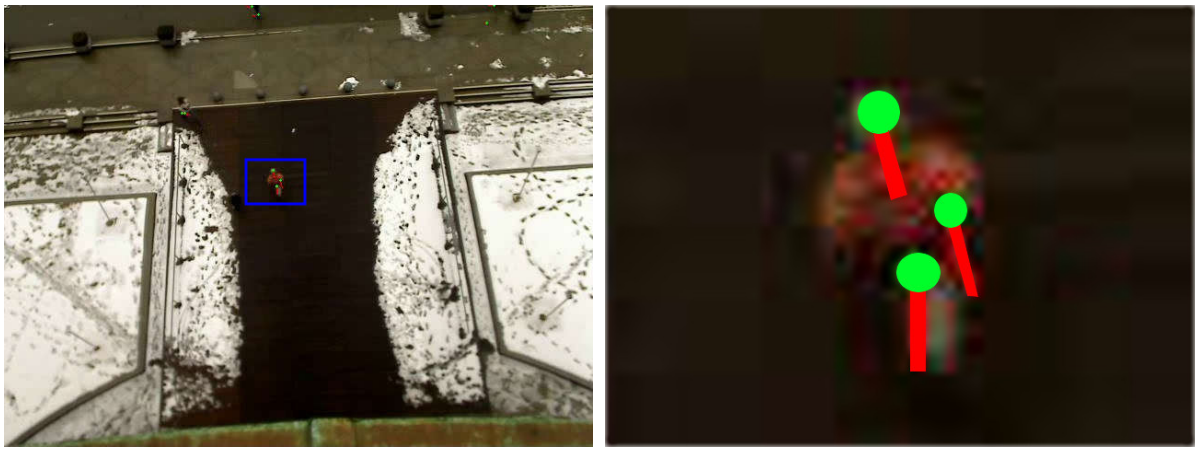
Chapter 3

Crowd Analysis

Crowd analysis plays two important roles in the proposed framework: 1) crowd analysis processes the exemplar (crowd) video and extracts information used during crowd synthesis and 2) crowd analysis is used to extract motion information from renderings of virtual crowds, which is used to ascertain the quality of crowd animation. Crowd analysis extracts dominant motion paths from crowd videos. Our method does not assume that individuals seen in the videos can be reliably tracked. Rather our method aggregates motion pixel motion observed between successive keyframes to identify the dominant motion patterns. We noticed that high-framerate videos—i.e., videos recorded at 60 to 120 frames per second—exhibit very small motion between two successive frames. In order to achieve noticeable motion between two adjacent frames, we uniformly subsample crowd videos along the time axis. This process yields an ordered list of keyframes.

3.1 Inter-frame Motion Extraction

The inter-frame motion extraction stage takes in two adjacent (key)frames and returns a motion vector (u, v) for every pixel location (x, y) in the first image. A motion vector (x, y, u, v) simply states that pixel at location (x, y) in the first frame I_1 has moved to location $(x + u, y + v)$ in the second frame I_2. Motion vectors encode the motion



(a) (b)

Figure 3.1: Sparse inter-frame motion vectors for pixels corresponding to a single pedestrian (left). The right image shows a closeup view of the region bounded by the blue rectangle. Green circles indicates arrow heads.

between two frames. For example, for videos recorded from stationary cameras, motion vectors describe the motion of non-static objects present in the scene. We assume that exemplar crowd videos are recorded from stationary cameras. Motion vectors, therefore, extract the inter-frame movement of pedestrians present in the scene.[1] Motion vectors can be extracted between two frames using 1) optical flow or 2) feature tracking. Some methods extract a sparse set of motion vectors, meaning (u, v) is not computed for every pixel location. Feature tracking methods and sparse optical flow methods fall into this category [36]. It is also possible to extract a dense set of motion vectors, which compute a (u, v) for every (x, y) location [9]. Figure 3.1 shows a sparse set of motion vectors encoding the inter-frame movement of pixels corresponding to a single pedestrian. Below we briefly discuss optical flow based and SIFT based methods for computing inter-frame motion vectors. Technical details about these methods are readily available in many computer vision textbooks, such as [49].

[1]This, of course, is a simplification. The motion vectors will capture any movement perceived in the image. Its just that our videos only contain crowds moving against unchanging backdrops.

(a)

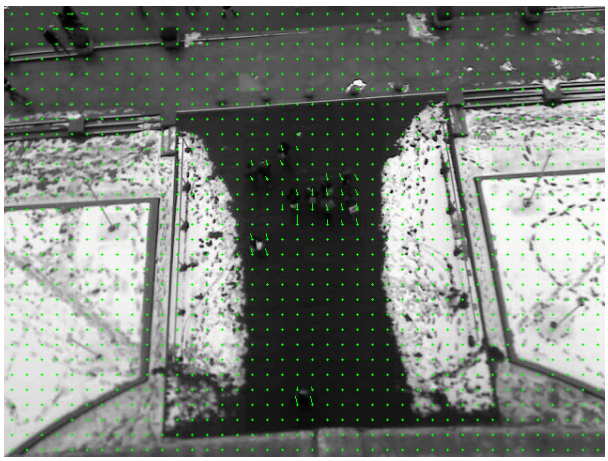
Figure 3.2: Showing sparse optical flow between frames 1200 & 1203 on the campus video using the Lusac-Kanade method.

3.1.1 Sparse Optical Flow

The Lucas-Kanade method [29] of optical flow looks at the neighboring area of each pixel (under the assumption that neighbors exhibit similar motions) to determine how the pixel moves between sequential frames. This is accomplished using least squares. Similar to SIFT, image pyramids, are used to observe motion at varying scales. Images pyramids are constructed from the repeated smoothing and subsampling of an image. Per pixel Lucas-Kanade is a dense optical flow solution. To perform sparse optical flow, keypoints are found and the Lucas-Kanade method is applied to those keypoints as opposed to each pixel. This performs better than the dense solution as less points are considered (Figure 3.2). The sparse Lucas-Kanade approach was found to be ideal in that it was the fastest and provided a sufficient number of flow vectors for performing dominant motion extraction.

3.1.2 Dense Optical Flow

The Farneback method [9] provides dense optical flow. Dense is indicative of a per pixel optical flow solution (Figure 3.3). Each pixel is given a flow vector representing



(a)

Figure 3.3: Showing dense optical flow between frames 1200 & 1203 on the campus video using the Farneback method. Every 8th pixel is shown.

the motion estimation between frames. This is noticeably different than the SIFT and Lucas-Kanade methods which are simply providing motion flows where motion occurs. Dense optical flow solutions excel in situations with a lot of motion such as moving cameras. Farneback's solution specifically provides two-frame motion estimation based on polynomial expansion. The output of Farneback's solution is a 2-channel array containing the per pixel optical flow vectors.

3.1.3 Scale-invariant feature transform

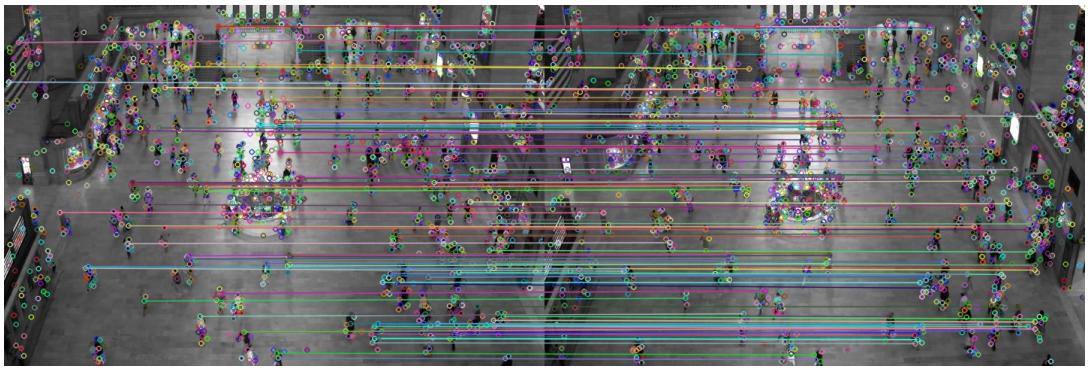
SIFT method [28] extracts local feature descriptors from an image. It has many applications in computer vision and is used here as a method of extracting motion vectors between two frames (Figure 3.4). SIFT can be used to extract flow vectors by analyzing sequential frames and observing how keypoints move.

3.1.4 Motion Vectors: Observations

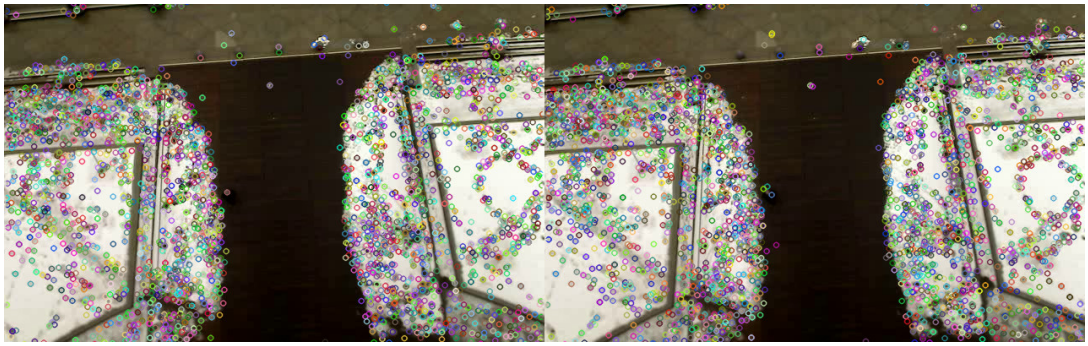
We have found Lucas-Kanade sparse optical flow to be the ideal tracking method for our purposes. Lucas-Kanade sparse optical flow provides less motion vectors than the

(a)

Figure 3.4: Showing extracted SIFT motion vectors between frames 1200 & 1203 on the campus video.



(a)



(b)

Figure 3.5: Compares SIFT performance in both scenes. Circles show key points while lines show matches between sequential frames. (a) is the grand central video. Here pedestrian motion accounts for much of the SIFT key point locations. (b) is the campus data set. Most of the key points for this video are none motion areas. SIFT does not perform well on this video as can be seen in Figure 3.6

.

Video	Method	Frames	Times	Vector Count
Grand Central	Sift	1380	15min40sec	45838
Grand Central	Dense	1380	5min39sec	22405680
Grand Central	Sparse	1380	33sec	430533
Campus	Sift	1500	21min59sec	10919
Campus	Dense	1500	5min06sec	17055000
Campus	Sparse	1500	32sec	10600

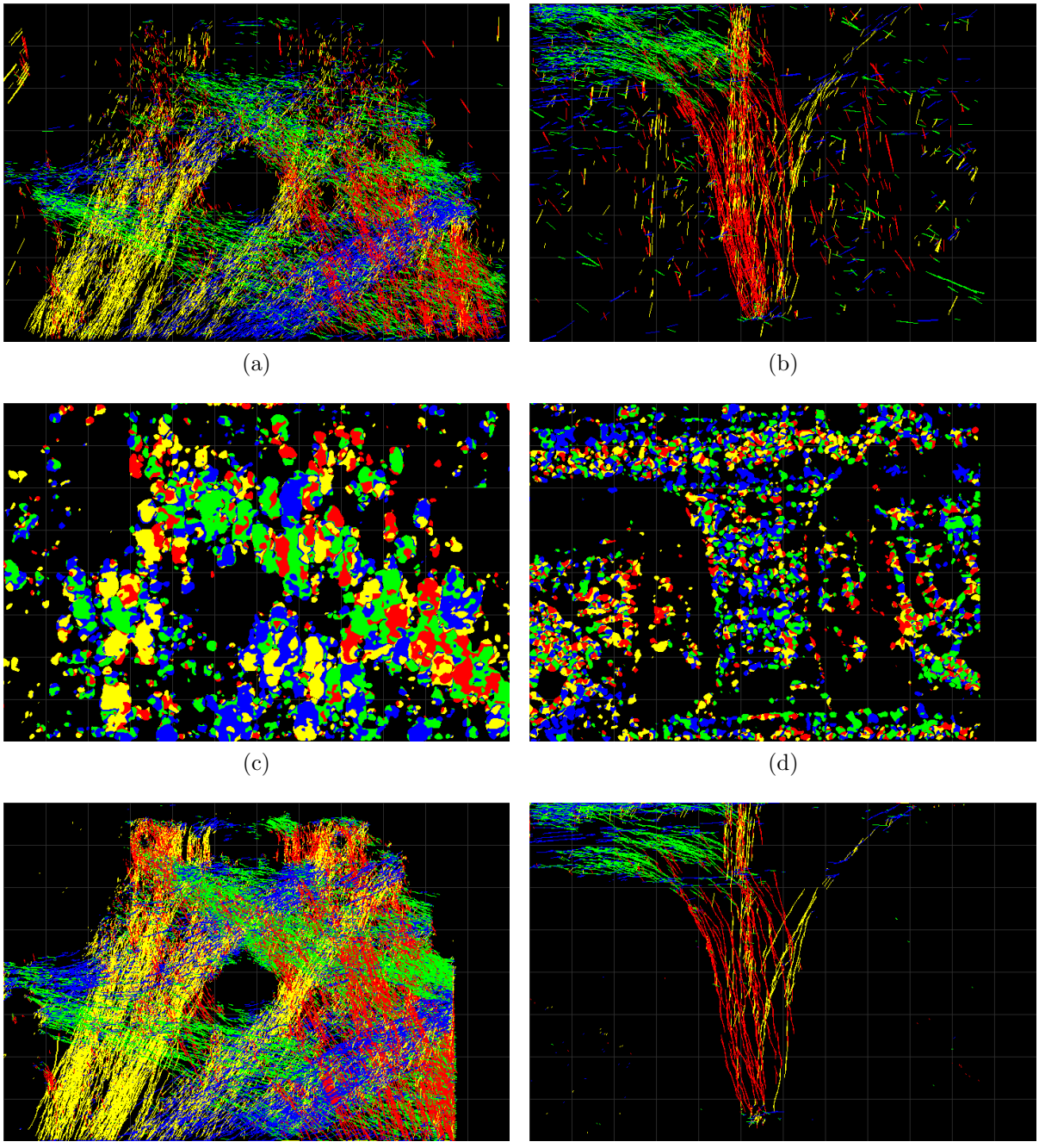
Table 3.1: 60 seconds worth of frames were processed for each video to compare the performance of the three motion vector extraction methods. Every third frame is used and small vectors are removed through thresholding. Resulting images in Figure 3.6.

Farneback dense optical flow while still tracking pedestrians within a given scene while working on both real and synthetic video. Through experimenting (Figure 3.6) with different tracking techniques we found the SIFT feature, as used by Ozturk et al., failed to observe and perform as well as Lucas-Kanade's method. Table 3.1 shows the performance of the three methods. SIFT needs to be fine-tuned to a video while the dense and sparse optical flow approaches work better in the general case. Between the optical flow approach, they performed similarly with sparse requiring less time and resulting in less vectors (but not too few). As such, the Lucas-Kanade method is best for our application.

3.2 Motion Vector Clustering

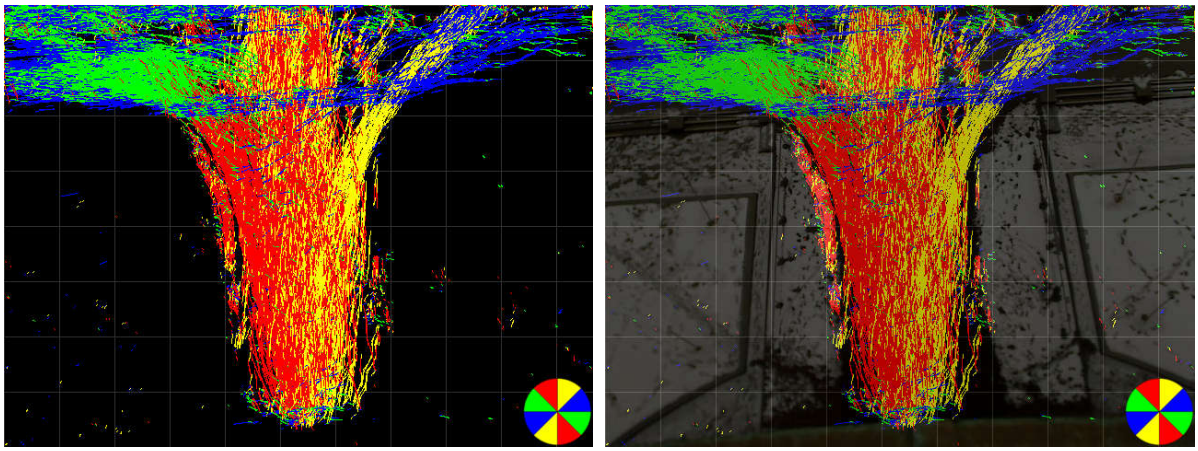
Given a sequence of (key)frames $I_1, I_2, I_3, \cdots, I_n$, inter-frame motion extraction returns a set of motion vectors (x, y, u, v, t), where t refers to the frame id and $t \in [1, n-1]$. We store these motion vectors as (x, y, θ, l, t), where $\theta = \arctan\left(\frac{y}{x}\right)$ and $l = \sqrt{x^2 + y^2}$. This representation facilitates orientation-based grouping of motion vectors. Figure 3.7 show all motion vectors computed for a given exemplar video.

We are interested in combining these motion vectors to construct dominant paths. This is accomplished through clustering. The image space is divided into cells (Figure 3.7)—cell extents are defined in pixel locations. Motion vectors belonging to the same cell are aggregated in an 8-bin orientation histograms $H_\theta^{(i,j)}$. (i, j) here refer to



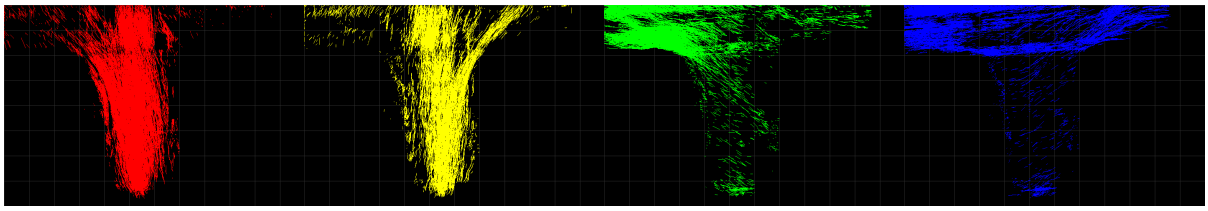
(a) (b) (c) (d) (e) (f)

Figure 3.6: Left side shows results on the grand central video. Right side shows results on the campus video. (a) and (b) are SIFT performance. (c) and (d) are dense optical flow performance. (e) and (f) are sparse optical flow performance. Note: vectors must have a minimum length to be included. Performance noted in Table 3.1.



(a) (b)

Figure 3.7: (Left) Motion vectors collected from a given video. (Right) Motion vectors overlaid on the first frame of the video. The motion vectors are color coded according to the color-orientation wheels shown in the bottom right corners. The overlaid grid indicates spatial binning. The 640×480 image is divided into 40×40 equal size cells.



(a) (b) (c) (d)

Figure 3.8: Motion vectors color-coded according to their orientation bin membership. Motion vectors with $\theta \in [90, 135]$ degrees and $\theta \in [270, 315]$ degrees are shown in (a). Motion vectors with $\theta \in [45, 90]$ degrees and $\theta \in [225, 270]$ degrees are shown in (b). Motion vectors with $\theta \in [135, 180]$ degrees and $\theta \in [315, 360]$ degrees are shown in (c). Motion vectors with $\theta \in [0, 45]$ degrees and $\theta \in [180, 135]$ degrees are shown in (d).

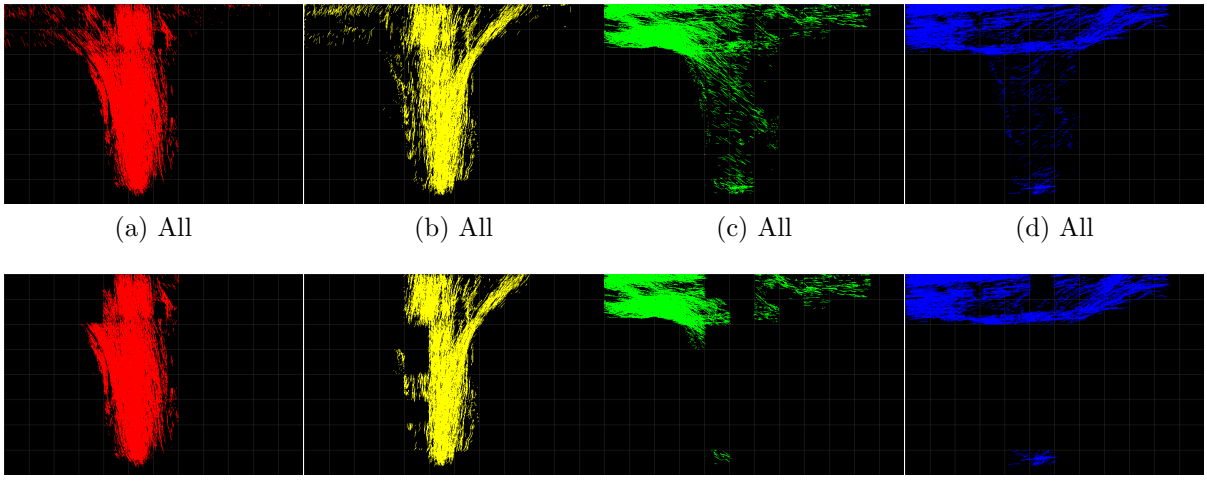
the location of the spatial bin—in the example shown in Figure 3.7, $i \in [1, 40]$ and $j \in [1, 40]$—and $H_\theta^{(i,j)}(k)$ refers to the k^{th} bin of this histogram, where $k \in [1, 8]$. After this step each cell is represented by an 8-bin orientation histogram. Figure 3.8 shows motion vectors from Figure 3.7 color-coded according to their orientations. Vectors belonging to diagonally opposite bins in the orientation wheel are shown in the same color. Following the work of Ozturk et al. [36], 8-bin orientation histograms yield acceptable results. However, it is straightforward to change the number of bins. Aggregating nearby motion vectors into orientation histograms has a desirable side-effect. It allows us to discard motion vectors that fall in orientation bins with little support, i.e., if the number of motion vectors in a particular orientation bin is less than a threshold, we can safely ignore that direction (for that spatial location) in subsequent processing (see Figure 3.9).

3.2.1 Spectral Clustering

Motion vectors within orientation histogram bins that survive pruning are then clustered to compute spatially locally dominant directions (See Figure 3.11). We use the Self-Tuning Spectral Clustering scheme proposed by Zelnik et al. [56]. The affinity matrix is computed as follows:

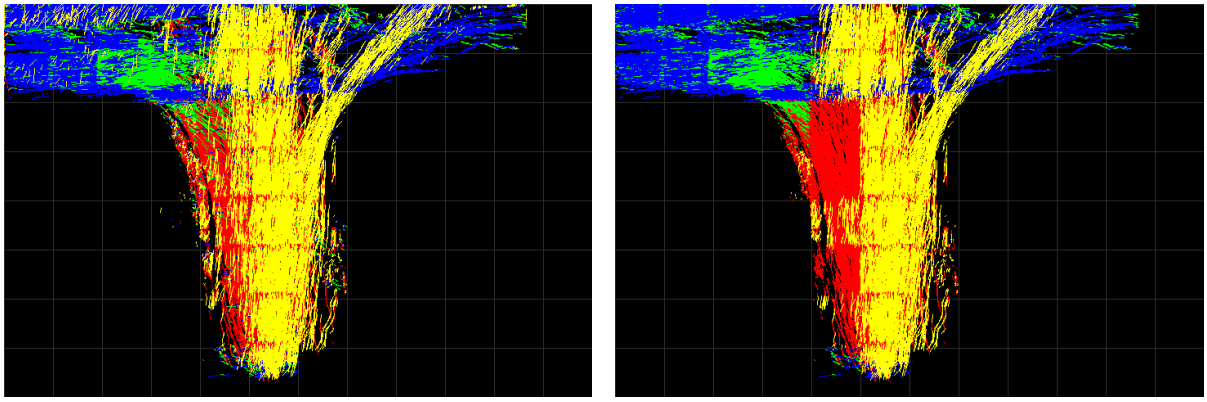
$$A(m, n) = \exp\left(-\frac{|\mathbf{p}_m - \mathbf{p}_n|^2}{\sigma_m \sigma_n}\right),$$

where $\mathbf{p}_m$ and $\mathbf{p}_n$ represent spatial locations (x, y) of the m^{th} and n^{th} motion vectors in bin $H_\theta^{(i,j)}(k)$. σ_m and σ_n represent scale values. Specifically, σ_m is the Euclidean distance between $\mathbf{p}_m$ vector and its k^{th}-nearest neighbour (in the same orientation histogram bin), and σ_n is the Euclidean distance between $\mathbf{p}_n$ vector and its k^{th}-nearest neighbour. Following Zelnik et al. advice, we use the 7^{th}-nearest neighbour when computing these values. The details of this algorithm are found in Zelnik et al. [56]. Clustering yields spatially local dominant directions (x, y, θ, w), where (x, y) represent the position, θ denotes the orientation, and $w \in [0, 1]$ indicates the weight (or support) for that direction (Fig-



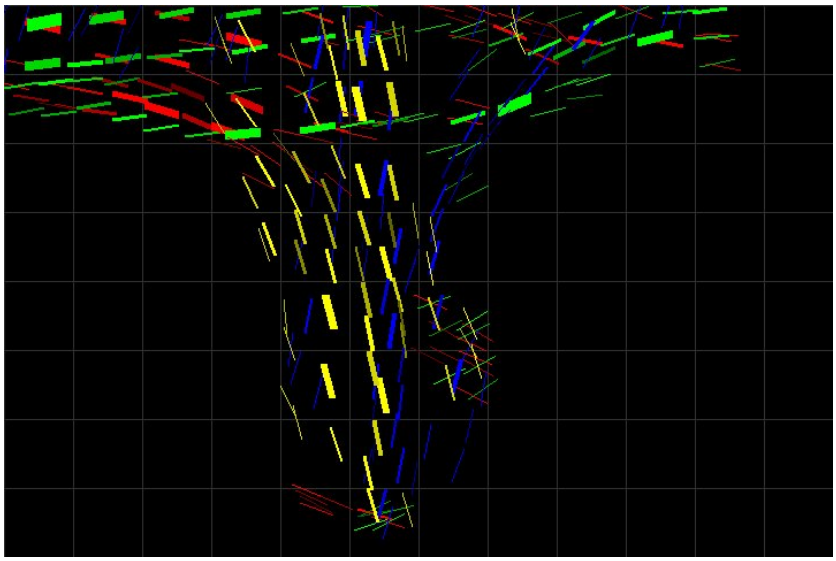
(a) All (b) All (c) All (d) All

(e) Discarded ($< 15\%$) (f) Discarded ($< 15\%$) (g) Discarded ($< 15\%$) (h) Discarded ($< 15\%$)



(i) All (j) Discarded ($< 15\%$)

Figure 3.9: Discarding motion vectors with low counts in their orientation bins. For this figure if an orientation bin has less than 15% of the overall mass, all motion vectors belonging to that bin are discarded. Again, the motion vectors are color-coded.



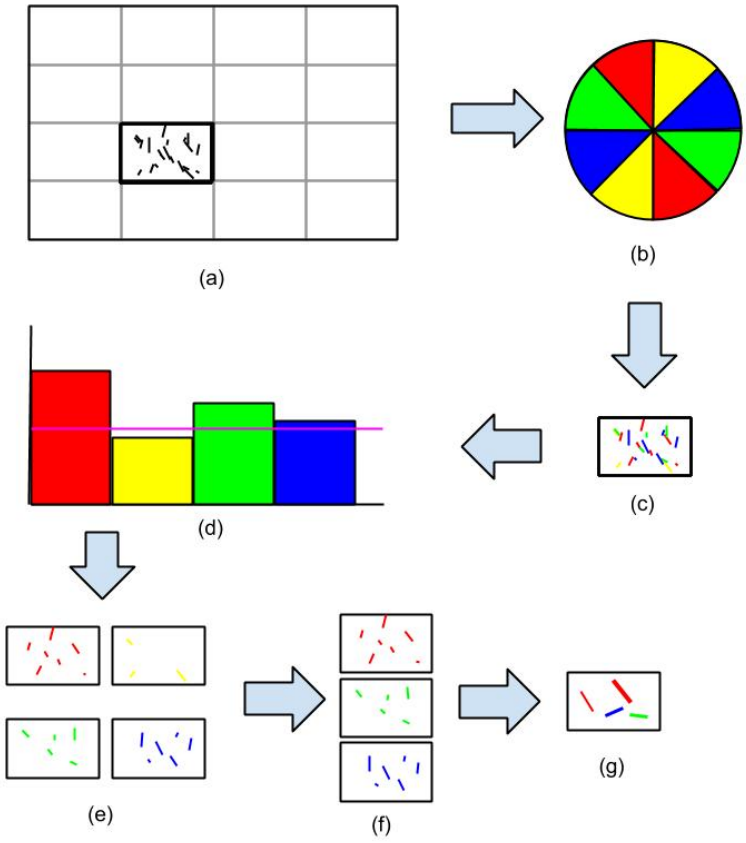
(a)

Figure 3.10: Spectral clustering returns spatially local dominant directions. The color of the vector represent its orientation; where as, its thickness indicates the number of motion vectors that belong to this cluster.

ure 3.10).

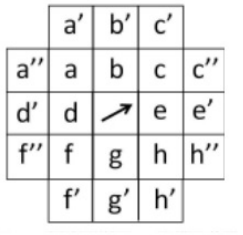
3.3 Path Generation

The next step is to combine these spatially local directions into global paths using the approach described in Ozturk et al. [36]. The basic idea is similar to contour grouping. Given a (dominant) direction vector, search in its neighbouring cells to find vectors having similar orientations and group the two vectors to grow the path. If the neighbouring cells do not contain any vector with similar orientation then consider vectors in other orientations. Figure 3.12 illustrates how neighbouring cells are identified for a given vector. All else being equal, directions with higher weights are given precedence. The details of this scheme are available in [36] and Section 7.3. In practice the cells are swept from left-to-right and from top-to-bottom to collect dominant direction vectors into global paths. Figure 3.13 illustrates global paths generated from dominant directions returned by the spectral clustering procedure.



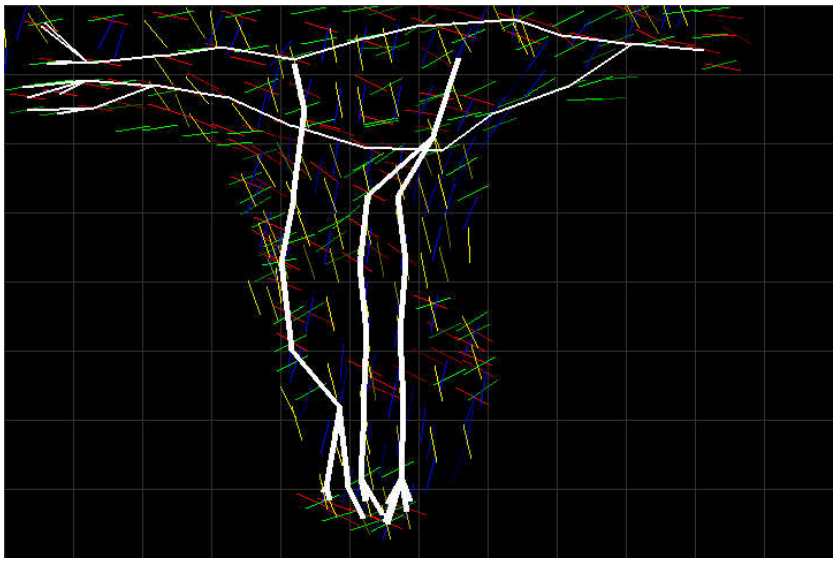
(a)

Figure 3.11: Showing the stages of clustering. (a) The scene is divided into cells. (b) motion vectors belong to each cell are assigned to one of 4 orientation clusters (demonstrated with one of 4 colors). (c) showing a single cell and it's motion vectors. (d) orientation clusters per cell are used to generated a histogram of orientations. If a orientation bin contributes less than threshold (pink line shown), that orientation is discarded. (e) shows a single cell and its per orientation vectors. (f) showing orientations for a single cell that are kept. These enter self-tuning spectral clustering. (g) The resulting locally dominant directions.



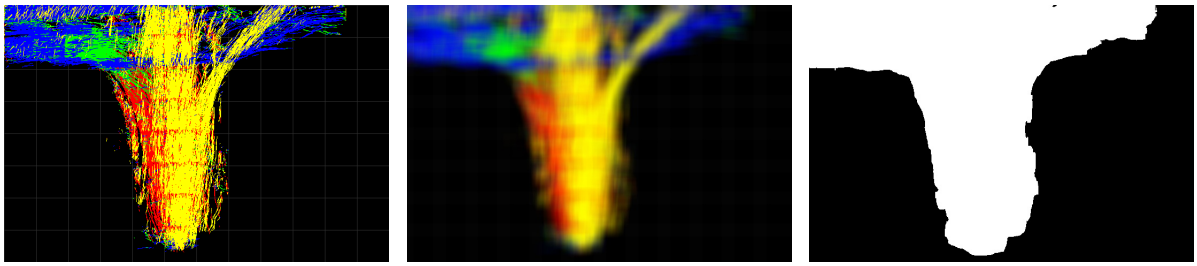
(a)

Figure 3.12: Demonstrates how neighboring cells are determined. Courtesy of [36]. All blocks with a common letter represent a single direction used in the determination of neighboring cells.



(a)

Figure 3.13: Resulting globally dominant paths shown in white.

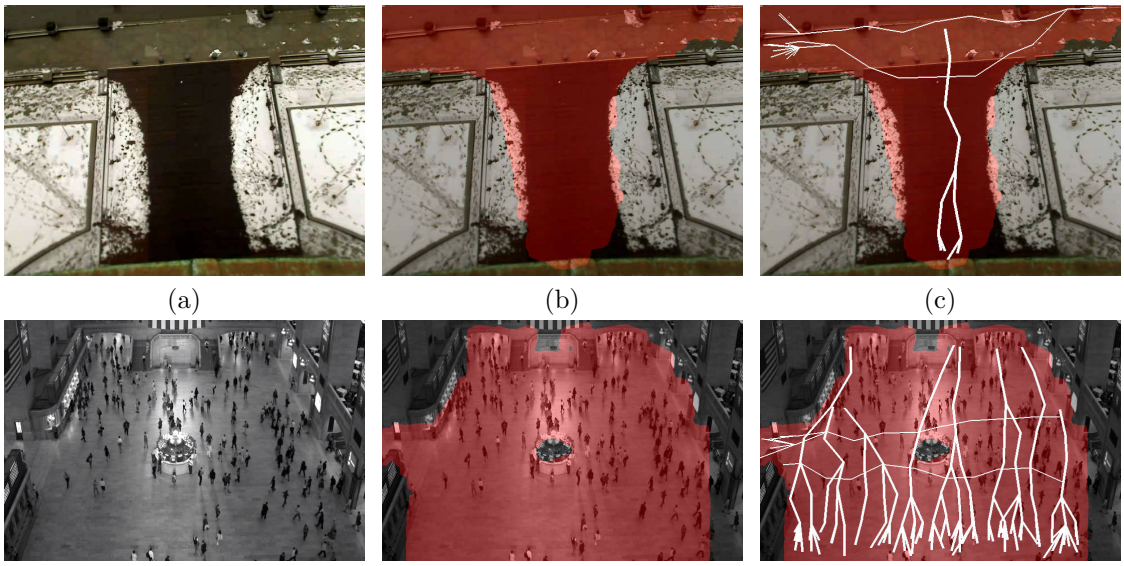


(a) (b) (c)

Figure 3.14: (a) All motion flow vectors (b) Blurred motion flow vectors (box filter with a width and height of 25) (c) resulting mask of the blur (black should not be navigated)

3.3.1 Motion Mask

Motion masks are binary masks which indicate areas of activity versus areas of inactivity. We gather motion vectors into a single frame to find all pixels belonging to motion. In doing this we are left with two types of pixels, those which have motion and those that do not (Figure 3.14). However, if we were to just use the image combining all motion vectors there would be many holes. To compensate for this we perform smoothing using a normalized box filter. The box filter simply takes a given pixel and replaces its value with the average of its neighboring pixels. A box filter observing the surrounding 25 pixels in the x and y dimension is found to work best. This helps to better define areas where we allow motion versus those where we do not (Figure 3.15). This information is used by the path generation and agent path planning logic to ensure agents do not receive paths or navigate from paths to obstructed areas.



(a) (b) (c)

(d) (e) (f)

Figure 3.15: Top row, (a)-(c), showing the campus scene alone, the scene with binary mask overlaid (red), and finally the scene with binary mask and resulting paths respectively. Bottom row, (d)-(f), showing the grand central scene alone, grand central with binary mask overlaid (red), and finally grand central with binary mask and resulting paths respectively. Note the fountain in the center is not navigable and the buildings on the sides are excluded.

Chapter 4

Crowd Synthesis

We now turn our attention to crowd synthesis. We use RVO2 behavior-based agent simulation system to simulate the movement of agents on a 2D plane. 3D virtual humans are animated along the trajectories returned by the RVO2 simulator. Human animations are driven by motion capture data within the Unity 3D game engine. We now describe various steps of this procedure.

4.1 Agent Simulation

RVO2 implements a behavior-based multi-agent simulation framework. Every agent is treated as an autonomous entity, complete with perception, decision-making, and action routines. Perception routines enable an agent to "observe" its environment, identify other agents, obstacles, and other items of interest. Perception plays a key role in intelligent and believable behavior of these agents. Without perception, an agent cannot make decisions that reflect the current state of its surroundings. Intelligent behavior presupposes some ability to perceive one's surroundings. There are also limitations on an agent's ability to perceive its surroundings. For example, an agent can only observe other nearby agents that are directly in front of it. Perception limitations give rise to believable behaviors. An agent that can perceive the whole environment, including every other agent, typically

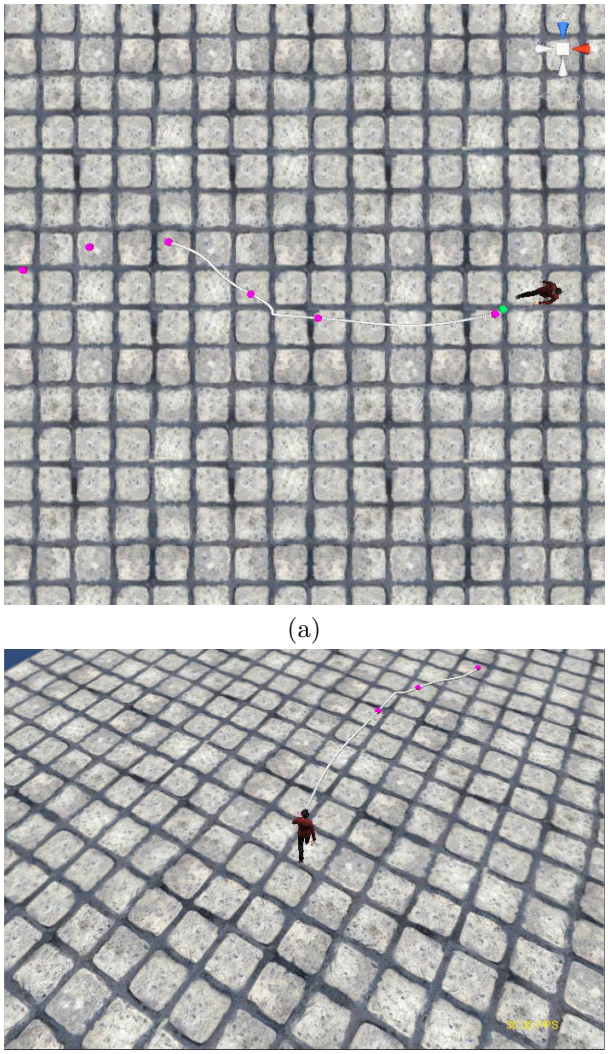
exhibit mechanistic movements.

Decision-making takes into account the current state of the agent, its surroundings, and its goals. It then updates the current position and velocity of the agent. Decision making has to contend with multiple, at times conflicting, goals of an agent. An agent might desire to move along a path with a certain speed. It is also poosible an agent may want to arrive at some location at a certain time. Furthermore an agent might be required to stop and avoid an impending collision. The interplay of these goals gives rise to the overall behavior of an agent. RVO2 implements start-of-the-art collision avoidance and goal arrival behaviors.

4.1.1 Goal Stack

Each simulated agent has a unique goal stack which contains the information needed to navigate the scene successfully. The goal stack will contain the nodes of an agent's desired path. When an agent is created, they receive their goal stack. The stack updates as each goal is met until there are no goals left to complete. When the goal stack is empty, the agent has reached the end of a path and can be removed from the system. Although the goal stack is used in our system purely for navigation, it can be used to support layered behavior.

Our agents also respond to subgoals. A subgoal can be pushed to the top of a goal stack to ensure an agent performs that task prior to completing their current goal. Subgoals are currently used in the system to achieve smoother agent trajectories. When an agent navigates strictly from one goal to the next, it results in very robotic behavior. An agent reaches a destination and promptly turns toward its next destination and begins advancing. Curve interpolation is used as a path smoothing method for agent navigation. Agents can receive subgoals which correspond to interpolated values along a curve connecting their goals. By navigating the subgoals, the agent is able to advance toward their goals with a more natural approach (Figure 4.1).



(a)

(b)

Figure 4.1: Showing agent navigation. Pink nodes shows the goals in the goal stack. The green node is the subgoal. The white path is the interpolated path for the agent. (a) Top view and (b) third-person view.

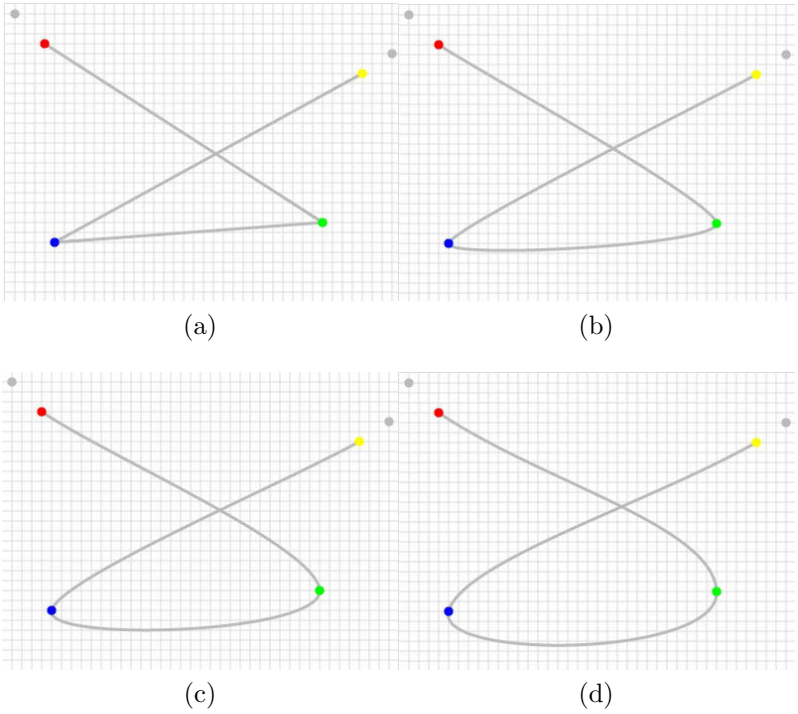
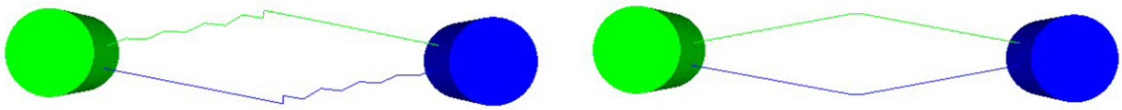


Figure 4.2: Demonstrating the tension parameter t. (a) $t = 0.0$ (b) $t = 0.33$ (c) $t = 0.66$ (d) $t = 1.00$

Path Interpolation

Path interpolation is performed by taking the desired paths and using the goals (nodes) to interpolate Cubic Hermite Splines along the path. Other curve fitting methods can of course be used, the cubic Hermite splines, however, give good results and ensure smooth continuous paths. This gives agents a more natural path to follow and avoids the situation where once they arrive at their current goal they immediately turn to face their next goal (Figure 4.2 (a)). Using this method the agents gradually turn toward their next goal as the pass their current one (Figure 4.2 (d)). Cubic Hermite splines have the added functionality provided by a tension parameter, allowing us to manipulate how strongly or weakly an agent sticks to a given path (Figure 4.2).



(a) (b)

Figure 4.3: Shows resulting paths of collision avoidance. (a) shows the result of the velocity obstacle collision avoidance approach. The oscillations can be seen in the jittery paths. (b) shows the result of reciprocal velocity obstacle collision avoidance. The paths are much smoother. Images courtesy of the Geometric Algorithms for Modeling, Motion, and Animation (GAMMA) research group [12].

4.1.2 Collision Avoidance

Collision avoidance among agents is performed using reciprocal velocity obstacles [52]. Reciprocal velocity obstacles is an extension of velocity obstacles which provides real-time multi-agent navigation and collision avoidance. Global agent navigation is managed with the goal stack; where as, reciprocal velocity obstacles are used for local collision avoidance.

Reciprocal Velocity Obstacles

The concept of velocity obstacles [10] involves determining how all agents should react to one another by determining where collisions will occur given agents maintain their current velocities. Each agent $a \in A = \{a_1, ...a_n\}$ has a position (x_a, y_a), radius r_a, and a velocity (u_a, v_a). Using this information, a velocity obstacle can be constructed. For a given agent, a_i, the collision cone with another agent a_j can be constructed. The collision cone is generated by adding the radius, r_{a_i} to r_{a_j} at position (x_{a_j}, y_{a_j}) (a_j's center of mass). The collision cone is generated with its apex at (x_{a_i}, y_{a_i}) and edges run tangential to the circle placed at (x_{a_j}, y_{a_j}) with radius $r_{a_i} + r_{a_j}$. By offsetting the collision

cone by (u_{a_j}, v_{a_j}), we are left with the velocity obstacle VO_{a_j}. Selecting a new velocity for a_i outside of VO_{a_j} will ensure a collision between the two does not occur. Selecting a velocity outside the velocity obstacles but also directed closest to a goal is typical. For many agents, the logic follows that determining all velocity obstacles and selecting a velocity outside of these obstacles will result in collision free navigation. Although this approach ensures collisions do not occur, there is an oscillation problem due to constantly adjusting agent velocities.

Reciprocal velocity obstacles [52] is a modification to velocity obstacles that overcomes the oscillation problem. The solution is a small adjustment to the selection of new velocities to avoid collision. Instead of choosing a velocity outside of the velocity obstacle (typically the closest aligned to the agents goal), instead select the average of the velocity outside the velocity obstacle and the current velocity. The result is a smoother and more natural collision avoidance.

4.1.3 Reciprocal Velocity Obstacle Simulation

RVO2 works by running a simulation of its own. For each animation cycle, RVO2 accepts an agent's position and a goal from its goal stack (typically a subgoal interpolated between goal nodes). This goal position is fed into RVO2 as the agents optimal destination. The RVO2 simulation performs its update, advancing the agent toward its goal while avoiding other agents and obstacles in the simulation. This returns the position where the agent will be at the end of the frame. This along with the agents current position provides a position delta which is used to compute the slice of animation performed by the animation driver.

We only use an instance of this simulation to perform the collision avoidance step. Every agent has its current position and velocity input as the initial step. Each agent also has a goal (from their goal stack) of where they would like to go. The agent's preferred velocity is calculated from the 2D vector of their current position and current

goal. The RVO simulation performs a single step which adjusts all agents positions and directions such that they proceed toward their respective goals without colliding with another agent. The result of this is the position the agent should be in at the end of the update cycle. The delta between the agent's current position and direction is fed to the animation driver (see below). RVO allows the user to specify a max speed at which agents can move. The simulation will slow agents down in situations where a collision needs to be avoided. The result of doing this is that the agents do not move the same distance with each simulation step. We need an animation system which is adaptive to these variable speeds. This is accomplished through the animation driver.

4.2 3D Human Animation

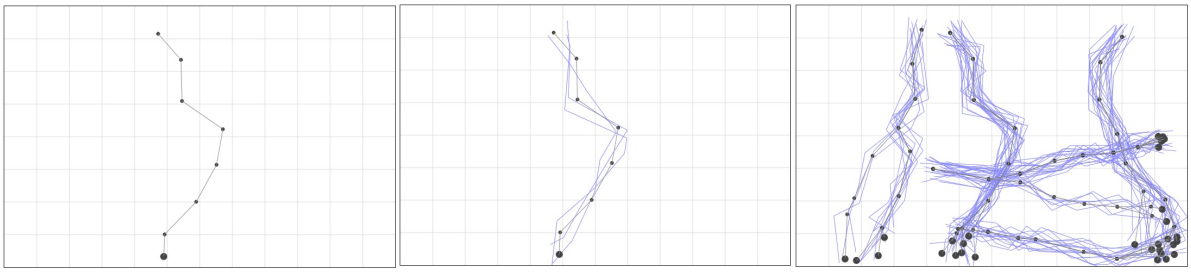
We simulate 3D human animations within the Unity3D game engine [51]. Unity3D is a powerful cross-platform game development platform, complete with advanced scripting, rendering, and animation capabilities. It is possible to import 3D virtual assets, including humans, objects of daily use, buildings, automobiles, and motion capture data within Unity3D, which greatly simplifies the task of creating computer animations. It is also possible to implement complex agent logic within Unity3D. We use the 2D trajectories computed by RVO2 to drive 3D virtual human animations within Unity3D.

The idea is as follows. Motion capture data provides animation snapshots for each agent. Stringing together multiple such snapshots creates the illusion of motion. The tricky part here is to control the speed of animation. Motion capture data is not available for all possible walking speeds. For example, say, we have motion capture data for a snapshot of "slow walk," "fast walk," etc. How do we create an animation that shows a human walking at a "medium speed." This problem has been studied within the computer animation community and Unity3D provides a *blending* mechanism to re-target recorded animations to achieve the desired result. Our system uses the position and

(a)

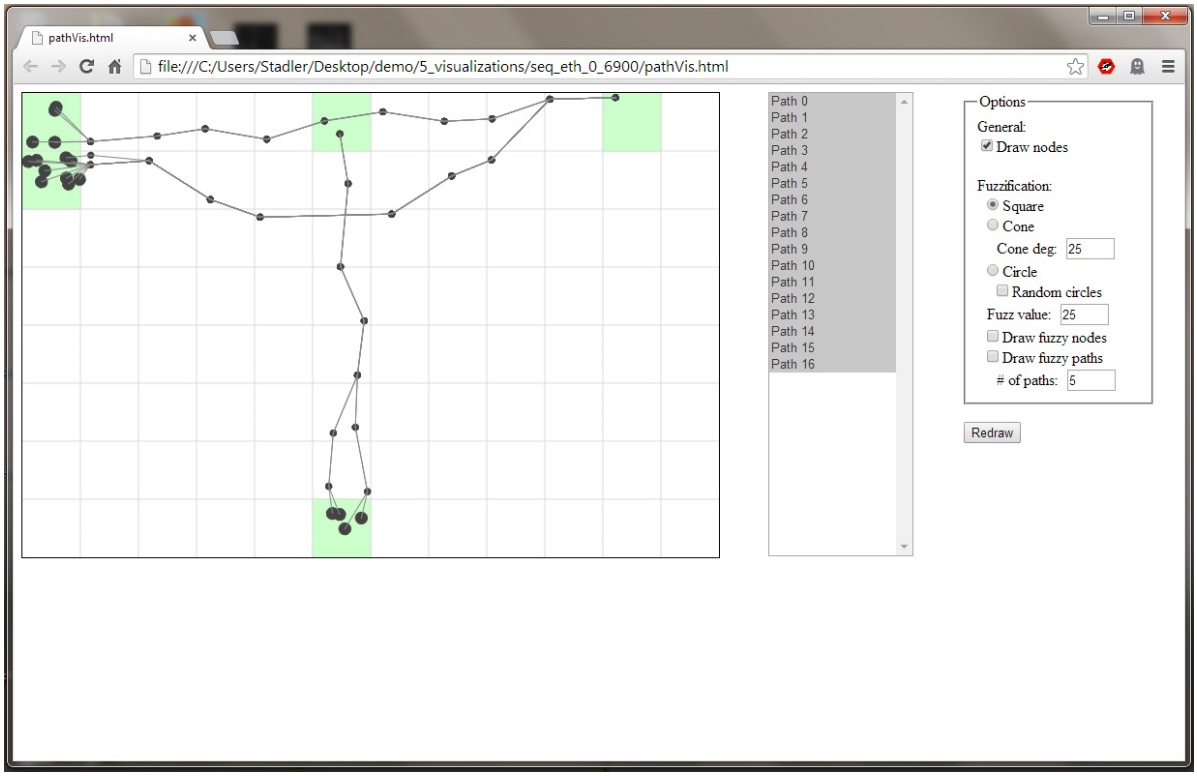
Figure 4.4: Shows agents following crude paths. Note the five agents in the top center area form a line. This is not common in crowds and jumps out as unrealistic. Viewers are able to spot how agents navigate the scene if they follow tight paths.

speed information returned by the RVO2 simulator and creates an animation snapshot of appropriate speed by blending the available motion capture snapshots.



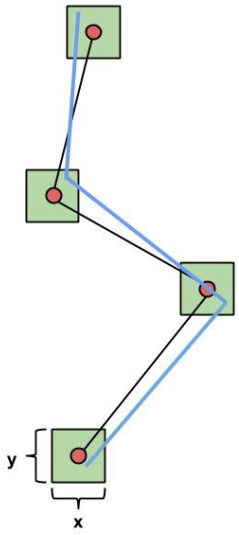
(a) (b) (c)

Figure 4.5: (a) Shows a single crude path (Note: larger node indicates the head, but the path can be navigated both ways). (b) Shows a single path after diversification. Diversified paths are shown in purple, there are 3 in this example. (c) shows all crude paths with 3 diversified paths overlaid.



(a)

Figure 4.6: (a) the visualization tool for playing with paths

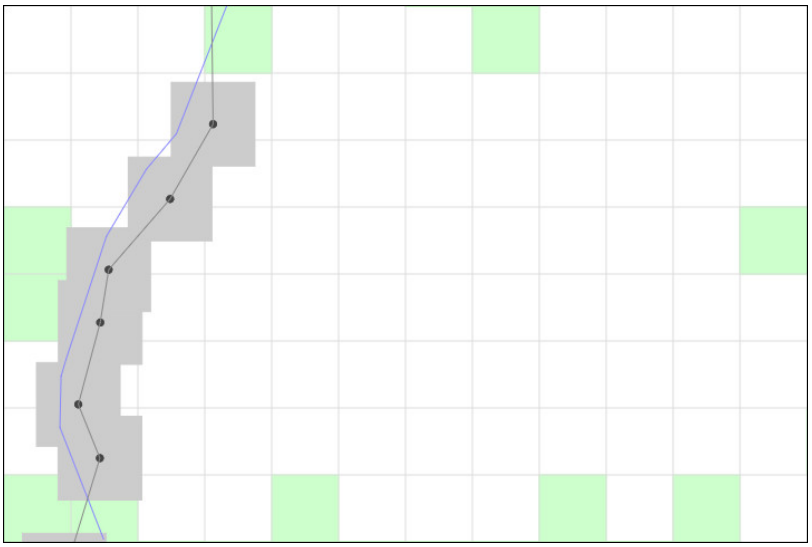


(a)

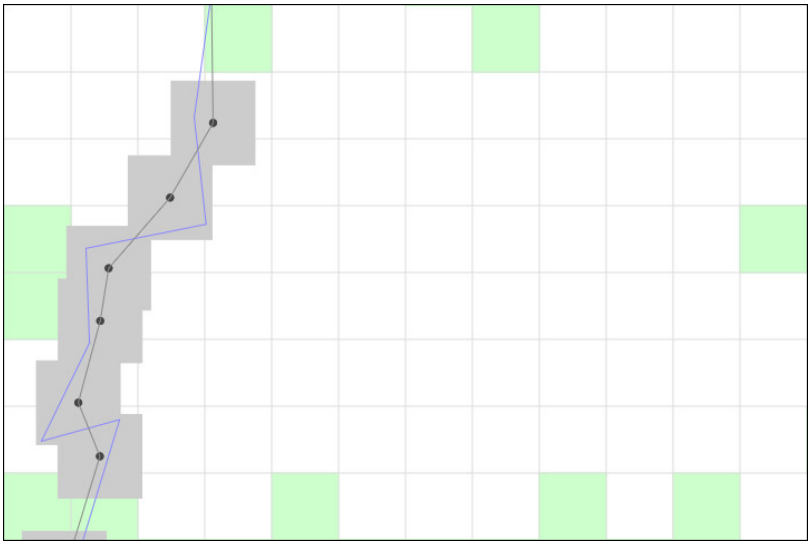
Figure 4.7: Demonstrating the square method of diversification. Red points are nodes in the original path. The original path is shown in black. The green squares represent the diversification area. A point is selected from this region. The blue path is an example of a diversified path.

4.3 Path Diversification

Crowd analysis returns dominant paths. If we instruct RVO2 to simulate agents that move along these paths only, the crowd simulation will exhibit an ant-like behavior: agents moving along invisble lines in the scene (Figure 4.4). This is undesirable. We solve this issue through path diversification (Figure 4.5), which is the process that generates multiple, slightly different versions of a given path. Given a single path through the scene, path diversification allows us to assign a unique path to each agent. We have experimented with three methods for generating variations from a given path. We call these methods: square method, triangle method, and circle method. We also developed a web-based interactive tool for experimenting with various path diversification strategies (Figure 4.6).

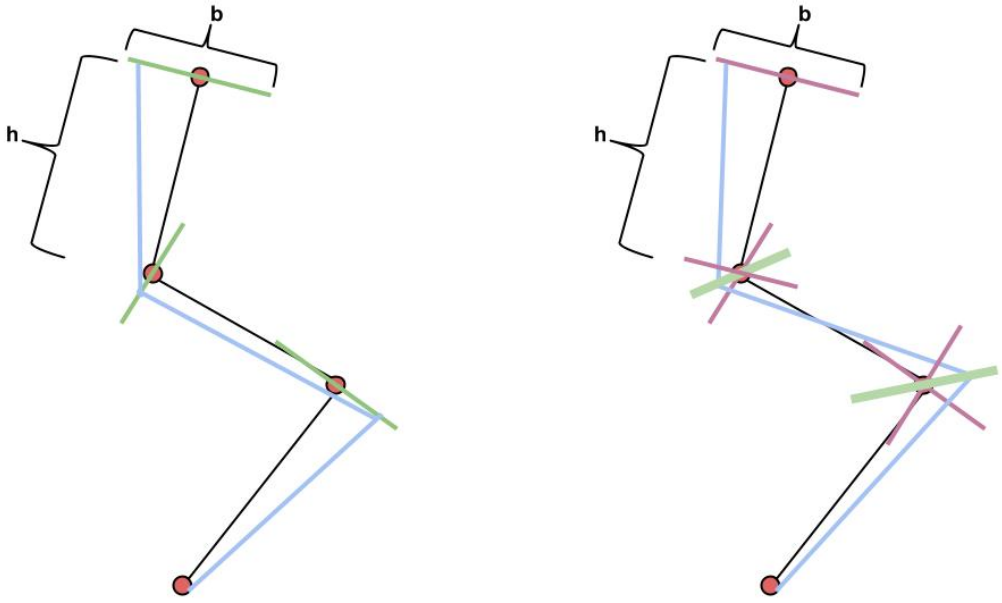


(a)



(b)

Figure 4.8: Showing results of the square method. (a) is an example of a good path produced using the square method. (b) is an example of a poor path generated by the square method. The poor path doubles back.



(a) (b)

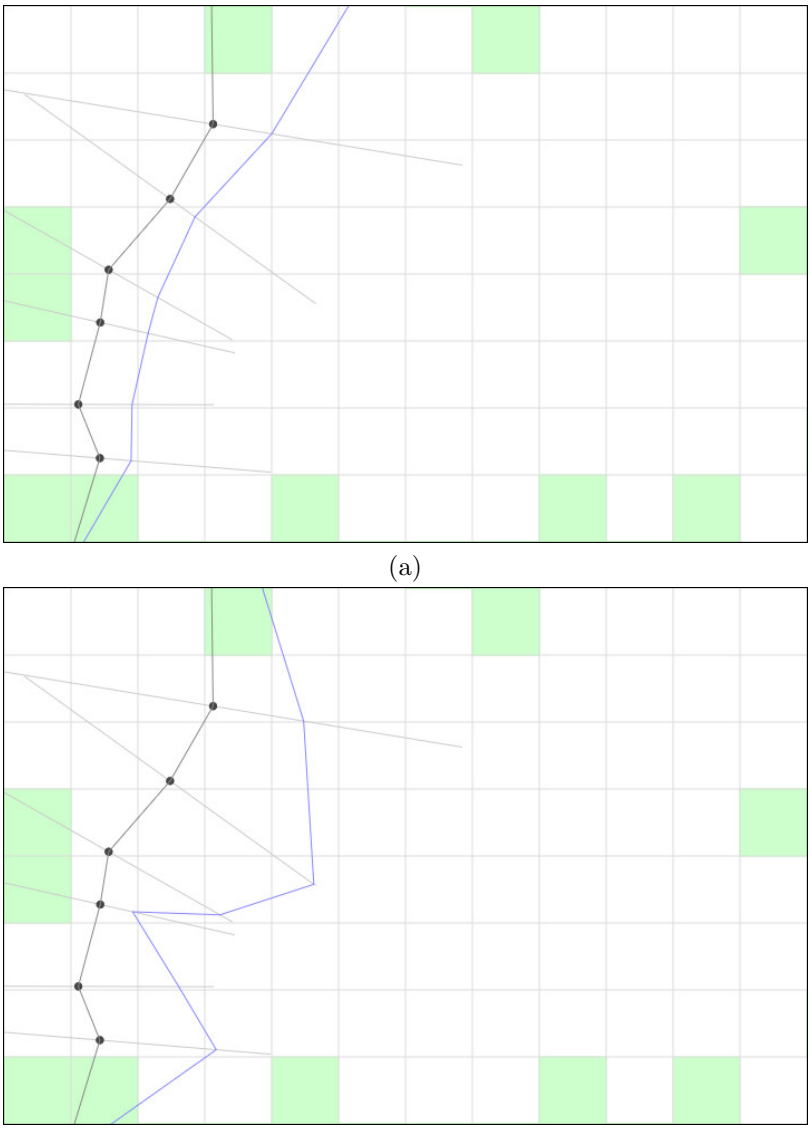
Figure 4.9: Demonstrating the triangle method of diversification. Red points are nodes in the original path. The original path is shown in black. The green lines represent the diversification area. A point is selected from this region. This region is a line which is aligned as the average of the perpendicular vectors to the two adjacent path segments. The blue path is an example of a diversified path using the triangle method. This method is referred to as the triangle method due to the base, b, being proportional to the length of the previous segment, h. Connecting the previous point to b creates a triangle.

Square Method

The square method simply makes a square area (user defined x and y) around each node and randomly selects a point from the area as the next node (Figure 4.7). This occurs for each node until a diversified path is generated (see Figure 4.8). This method suffers from overlapping areas when the areas are too large.

Triangle Method

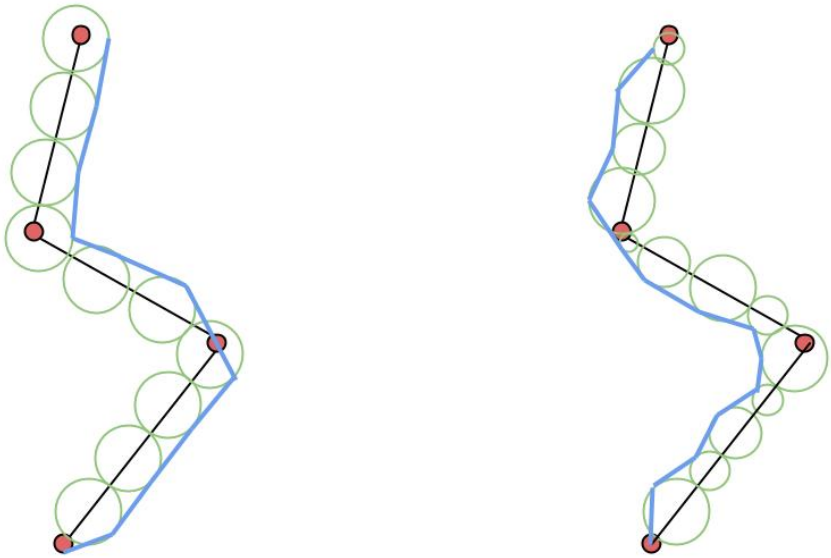
The second method of diversification is the triangle method (Figure 4.9). With this method the user defines the size of the base of a triangle drawn at the next node from the current node. The base is drawn as the average of perpendicular vectors of the



(a)

(b)

Figure 4.10: Shows results of the triangle method. (a) is a smooth path generated with this method. (b) is a poor path. The lower path has a huge dip.



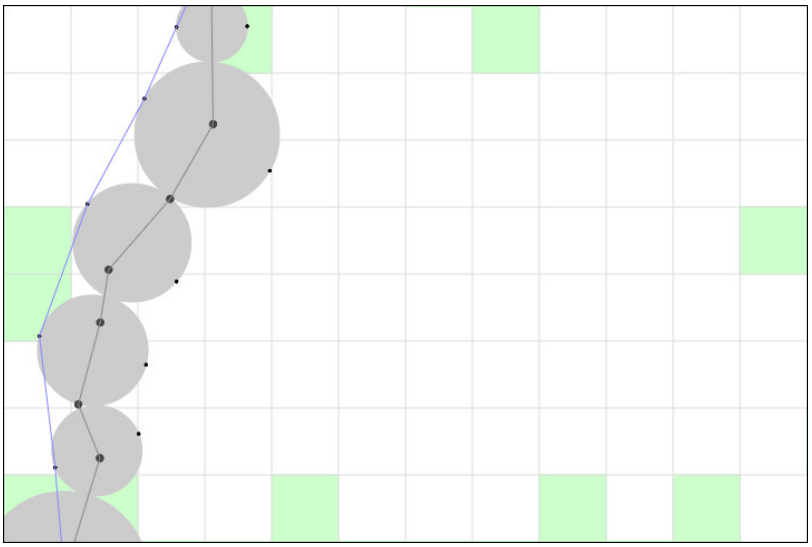
(a) (b)

Figure 4.11: Showing circle diversification. (a) image shows a path (blue) generated from circles of equal side. (b) image shows a path generated with circles of variable size.

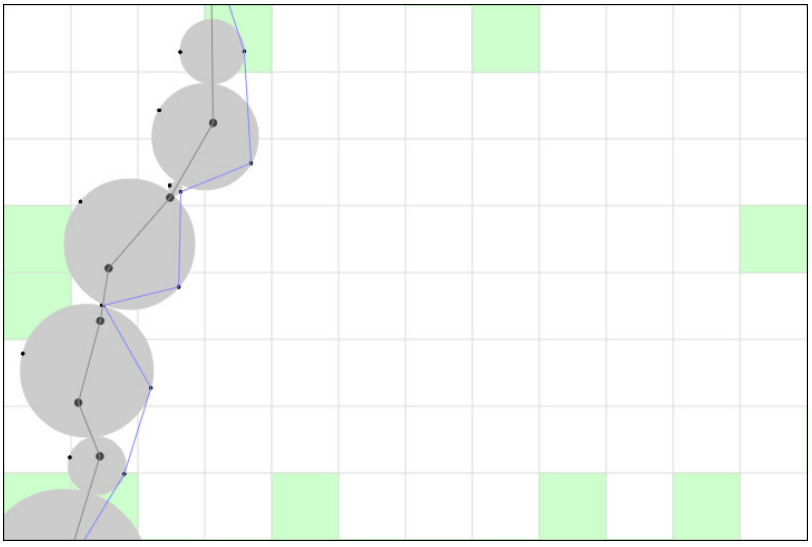
current and next path segments (see Figure 4.10). This method is less likely to suffer from overlap; however, it still occurs in some scenarios.

Circle Method

The last method we used for diversification is the circle method (Figure 4.11). In this method the user defines a maximum circle size and the algorithm randomly generates circles of max size or less along the path to the goal (see Figure 4.12). Sequential circles have related radii so the resulting paths do not have huge variations. This method has no overlap. It should be noted that the resultant path is randomly placed on one side of the crude path only. This logic was added to prevent the path crossing the crude path and then crossing back (very sporadic result).

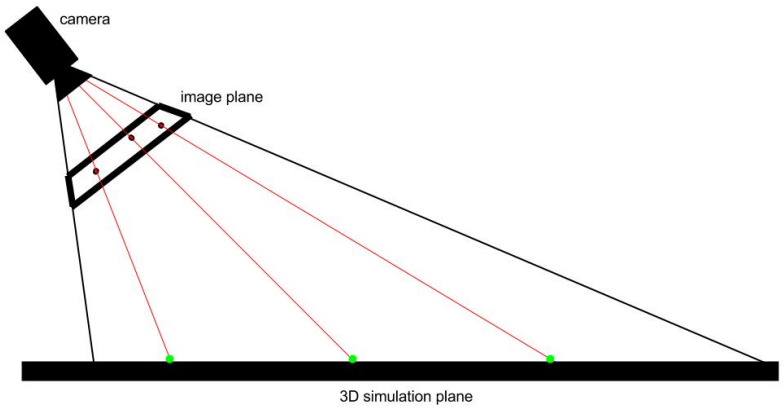


(a)



(b)

Figure 4.12: Shows results of the circle method. (a) is an nice path produced using the relative radius method. (b) is a poor path generated using simply random radii. The lower path is sporadic.



(a)

Figure 4.13: Showing how path nodes are projected into the 3D simulation plane. Rays (red lines) are cast into the scene from the camera position through path node pixels (red dots) on the image plane. The collision points (green dots) with the simulation plane are the new location of the nodes.

Some Thoughts on Path Diversification

The square method proposed has an issue with overlapping areas causing back tracking of agents or very sporadic behaviour for the agents. Similarly, although less common, the triangle method was able to produce this same backtracking and sporadic path diversification. The circle method was developed to resolve the backtracking problem noticed with the square and triangle methods. The circle method successfully prevents this issue; however, the sporadic behaviour was still observed with the circle method. To compensate for this, the sequence of circles generated was such that successive circles were generated with a radius relative to the previous circle. This prevented the sporadic behaviour and yielded a smoother diversified path. When diversified paths are used with reciprocal velocity obstacles they are navigated with gradual turns resulting in further smoothing of the motion. The circle method was found to work best and was used as the method of diversification.

4.4 From 2D to 3D and Back

Motion vectors extracted from the exemplar video live in the image space. Similarly global paths also live in the 2D image space. RVO2 agents also live in the 2D space. We are, however, interested in synthesizing 3D crowds. We achieve this by making a ground plane assumption: virtual humans only walk on a (2D) ground plane. This can be easily accompanied by back-projecting the global paths (and their variations generated through path diversification) onto the ground plane. Back-projection is easy if we know the location and orientation of the camera with respect to the ground. This information is sometimes available for an exemplar video. E.g., if this video is captured by a calibrated camera. In case this information is not available, we manually pick the most likely location and orientation of the camera by observing the exemplar video. Our ground plane assumption has an obvious limitation. We currently only handle crowds that move in a single plane. For example, our system cannot deal with crowds going up and down the stairs or moving on escalators. Similarly, our system is unable to extract meaningful paths from exemplar videos that show crowds at multiple levels.

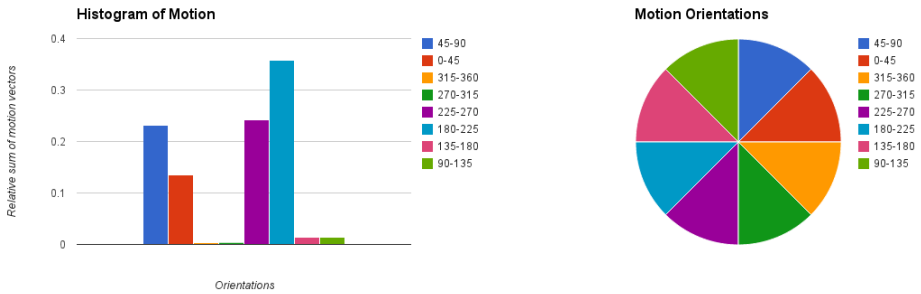
Unity3D allows us to render the crowd simulation into a video. We ensure that the rendered video has the same framerate and resolution as the exemplar video. Furthermore, for similarity computations the location and orientation of the camera used to record synthetic video should be as close to that of the camera used to record the exemplar video.

Chapter 5

Evaluation and Results

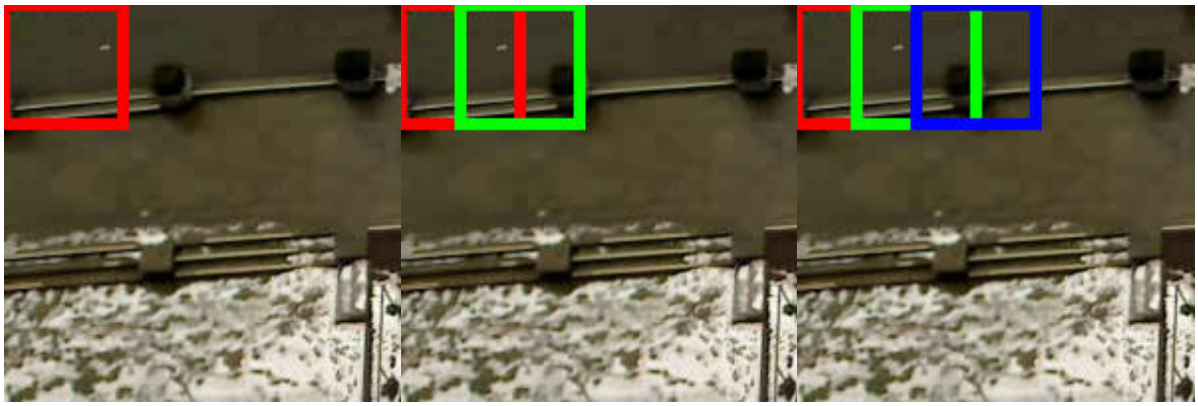
5.1 Evaluation

To ensure that our synthesized crowd is similar to the input crowd we need to perform a comparison. One straightforward scheme to compare the synthesized crowd with the crowd viewed in the exemplar video is to employ user studies. That, however, defeats the purpose of this work—we are interested in automated methods for synthesizing crowds from exemplar videos. Ideally our system will be able to replace user studies with a scoring system that can leverage image and video analysis for crowd comparison. This allows for iterative, self-tuning methods for crowd synthesis.



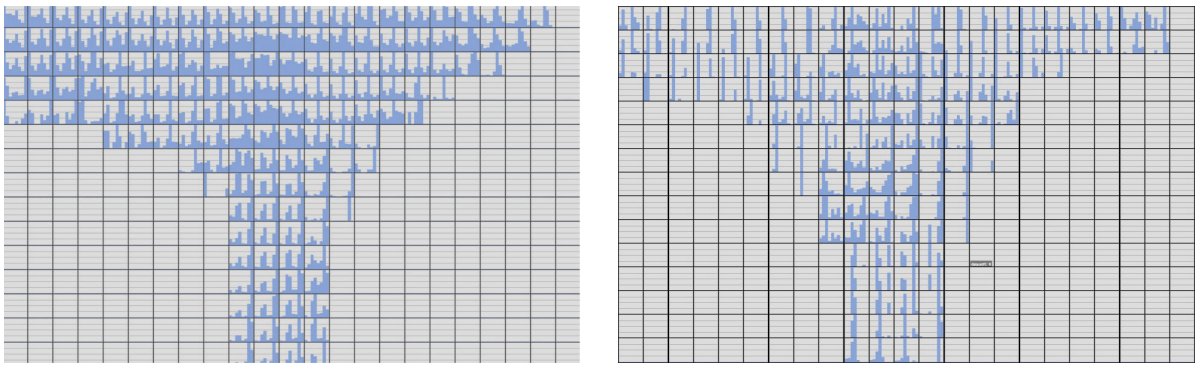
(a) (b)

Figure 5.1: (a) shows a histogram of motion for 8 motion vector orientation ranges. (b) shows the motion vector orientation ranges



(a) (b) (c)

Figure 5.2: Demonstration of the sliding window for histogram of motion generation. A small subsection (top-left) of the campus scene is used to demonstrate the 60x60 pixel sliding window. (a) shows the area of the first histogram of motion in red. (b) shows the area of the second sliding window in green. This overlaps the first window by 50%. (c) shows the area of the third histogram of motion. Notice it also has a 50% overlap with the second sliding window.



(a) (b)

Figure 5.3: Visualization output as a result of the histograms of motion generation. For this scene, histograms of motion were created using a sliding window of size 60x60 pixels. The window advances 30 pixels each iteration (50% overlap). (a) real Campus dataset video (b) synthetic result

Our scoring system is formed using histograms of motions. Histograms of motions show the distribution of motion directions for a given region in the image (Figure 5.1). In our system, the histograms of motions show the distribution of motion flow vectors using eight orientations. Recall that the first stage of the framework is to extract motion vectors. It has been found that sparse optical flow works best for both the real and synthetic crowd videos. Sparse optical flow is performed and the result is a collection of all motion flow vectors for the processed frames. These collections of motion flow vectors are what are being compared using the histograms of motions.

Instead of computing a single histogram of motions for the entire image, our method subdivides the image into rectangular regions and generates a series of histograms of motion directions. This is similar to how local bins are used to cluster flow vectors based on orientation and spatial location in the dominant path process. However, to remove discrete barriers between one histogram and the next, we utilized a sliding window to generate the series of histograms of motions (Figure 5.2). These histograms are normalized. For the remainder of this chapter, we assume that sliding window operation creates m histograms (i.e., unique sub-windows) for each video.

The system outputs a visualization of the histograms of motions (Figure 5.3). This visualization makes it easy to spot differences between scenes and how agents move through them. This visualization is good for us, but generating a relative score would prove even more useful for comparing histograms. The histograms between real video data and the synthesized crowd video data are compared using the Bhattacharyya distance. The Bhattacharyya distance measures the dissimilarity between two distributions (histogram), outputting a value between 0.0 and 1.0 with 0.0 meaning the histograms are a perfect match and 1.0 meaning the histograms are opposite. The Bhattacharyya distance is defined as:

$$d(H_1, H_2) = \sum_{i=1}^{n} \sqrt{H_1(i) \times H_2(i)},$$

where n is the number of bins and $H_1(i)$ and $H_2(i)$ are bin counts for i^{th} bins of histograms

H_1 and H_2. The final similarity score between the two videos is:

$$s(v_1, v_2) = \frac{\sum_{i=1}^{m} d(H_1^i, H_2^i)}{m},$$

where m is the number of histograms extracted for each video (through sliding window procedure), H_1^i and H_2^i are the i^{th} histogram for videos v_1 and v_2, respectively. From here we perform a series of tests and iterative modifications to the simulation to minimize this score.

5.2 Results

Video	Resolution	#Frames	Frame rate	Description
seq_eth.avi	640x480	12950	25fps	Bird's eye view of a campus walkway, the sides have yards which are blocked. These are obstacles and agents should not navigate here. Upper sidewalk goes left and right while center sidewalk goes up and down. Sparse crowd.
grandcentral.avi	720x480	46000	23fps	Looking down at Grand Central Station with a slight angle. Agents move freely in most directions. The center of the scene has an obstacle (fountain). Dense Crowd.
879-38.mov	480x360	1275	25fps	This UCF crowd video is a overhead view with a slight angle of people moving seemingly randomly.

Table 5.1: Describes the three video files used for experimental results.

Our framework is tested on 3 crowd videos, each with its own challenges and intricacies (Table 5.1). One video comes from the BIWI Walking Pedestrian Dataset, the second is the Grand Central Station Dataset, while the final is from the UCF Crowd Dataset. All three videos have an overhead view, although the Grand Central Station and UCF have a slight angle.

Each video is tested in twelve scenarios to see how the proposed metric performs. The crowds are tested with three different densities which are scene specific. Furthermore the crowds are tested with tight or loose goals. This is pertaining to how easily a goal is

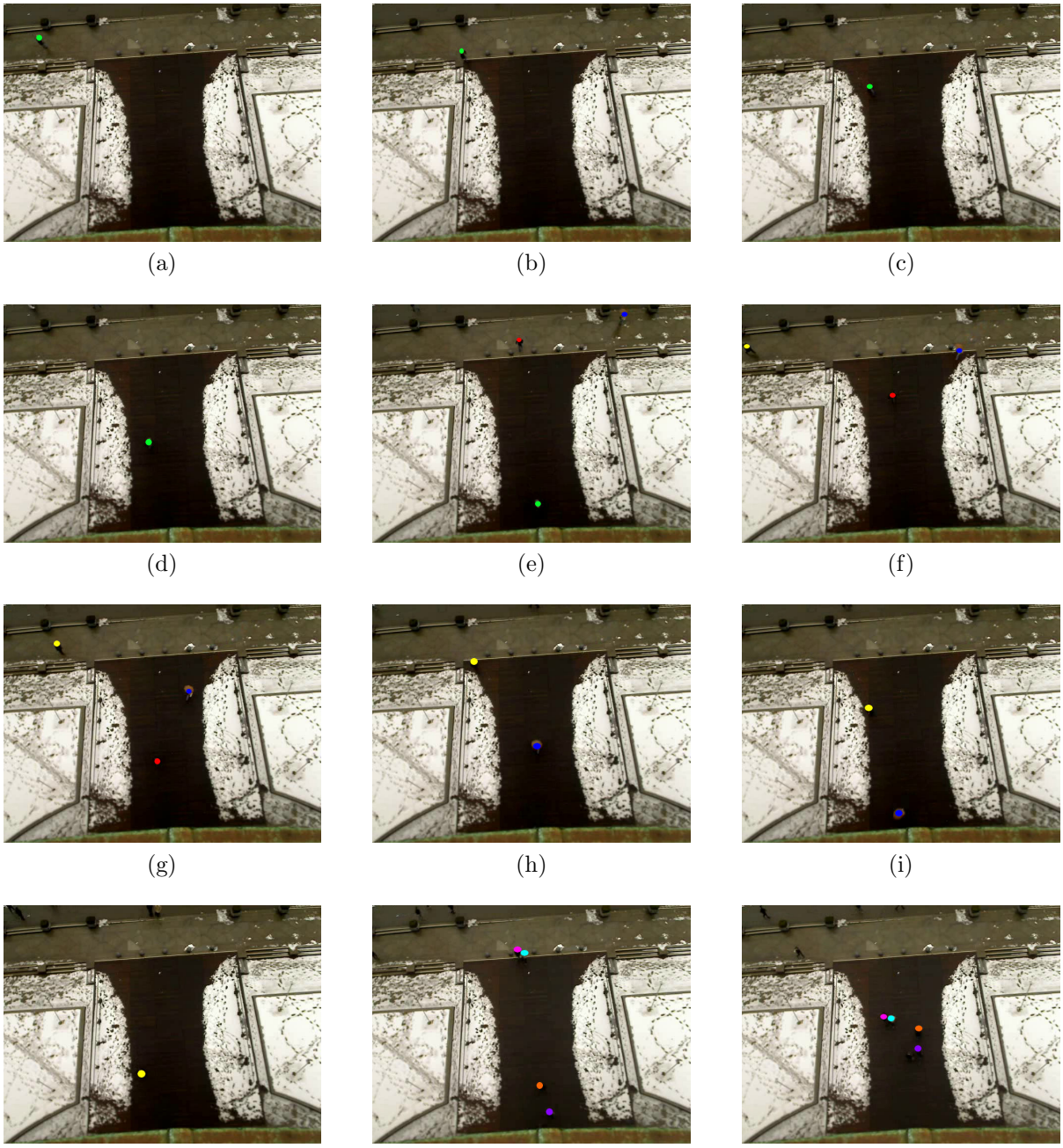
achieved. In the case of tight goals, a agent must be very close to the goal before they can advance, where as loose goals are a bit more forgiving. This was included as it generally causes problems with the natural flow of the crowd if goals are too tight. Agents tend to circle a goal or twitch if they are close to a goal and many agents are near. This scenario would easily been picked up by a human observer and labeled as unnatural. Lastly, the crowds are tested with and without path diversification. This test is performed because a human observer would be able to see agents forming single file lines. We test to see if our metric likes diversified paths or not.

5.2.1 Campus Video

Synthetic Crowd Characteristics	Score
05 agents, random paths	0.5506
10 agents, random paths	0.5080
15 agents, random paths	0.5033
05 agents, tight goals, no diversification	0.5462
10 agents, tight goals, no diversification	0.4970
15 agents, tight goals, no diversification	0.5001
05 agents, loose goals, no diversification	0.5487
10 agents, loose goals, no diversification	0.5120
15 agents, loose goals, no diversification	0.4619
05 agents, loose goals, diversification	0.5372
10 agents, loose goals, diversification	0.5210
15 agents, loose goals, diversification	0.4901

Table 5.2: Twelve synthetic crowds of the Campus Dataset and their scores based on Histograms of Motion.

Twelve synthesized crowds were generated from the campus videos. The random paths act as a test case, synthesized crowds should perform better than the random case. A population size of 10 agents was arbitrarily chosen as a starting point from the observed video, 05 and 15 are values selected relative to this. The results are shown in Table 5.2. The only characteristic that can be seen to consistently beat the random case on this dataset is using 15 pedestrians. The 15 pedestrian scenarios seem to produce the best scores, and best score overall, 0.4619, was accomplished with fifteen agents operating



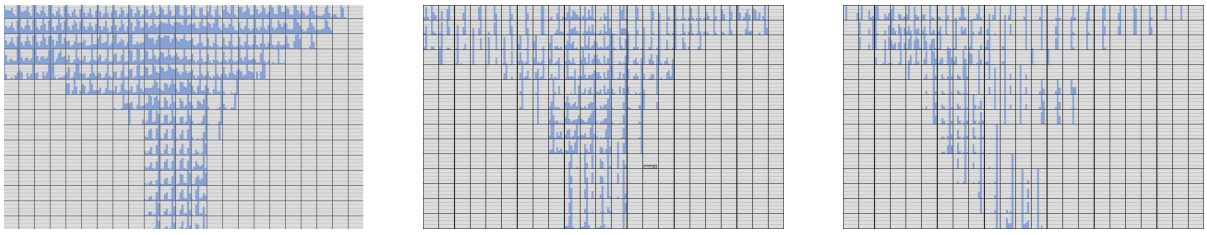
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.4: Shows frames from the campus video (original) demonstrating the type of motion present. It should be noted the campus video contains sparse crowds and is shot relatively overhead. Color labels added to some agents to assist viewing.



(a) (b) (c)

Figure 5.5: Campus video histograms. (a) Histograms of Motion from the real video. (b) Histograms of motion from the best (15 agents, loose goals, no diversification) synthetic crowd. (c) Histograms of motion from the worst (05 agents, random paths) synthetic crowd.

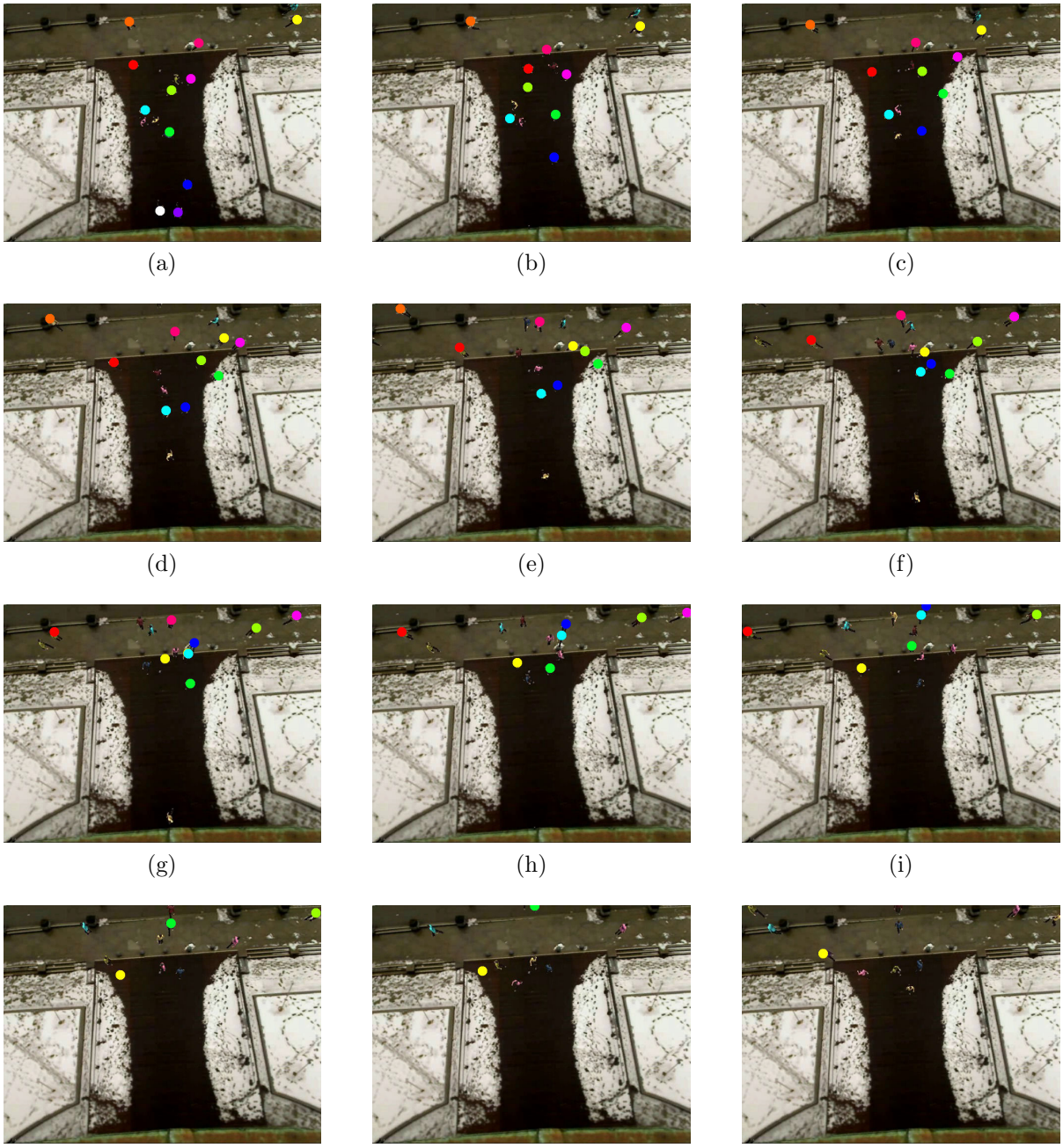
with loose goals and no path diversification. The worst score was found to be five agents operating on random paths, the score being 0.5506. The difference in the best and worst score is quite small.

5.2.2 Grand Central Video

Synthetic Crowd Characteristics	Score
50 agents, random paths	0.4539
75 agents, random paths	0.5048
100 agents, random paths	0.4421
50 agents, tight goals, no diversification	0.6091
75 agents, tight goals, no diversification	0.5790
100 agents, tight goals, no diversification	0.5677
50 agents, loose goals, no diversification	0.5844
75 agents, loose goals, no diversification	0.5845
100 agents, loose goals, no diversification	0.5808
50 agents, loose goals, diversification	0.4866
75 agents, loose goals, diversification	0.5624
100 agents, loose goals, diversification	0.4082

Table 5.3: Twelve synthetic crowds of the Grand Central Dataset and their scores based on Histograms of Motion.

Twelve synthesized videos were generated from the grand central video. The grand central video is denser than the campus video. There are many more agents present and they are observed from a angle. The crowd population sizes (50,75,100) in these tests are much larger than the other two data sets. The results can be seen in Table 5.3.



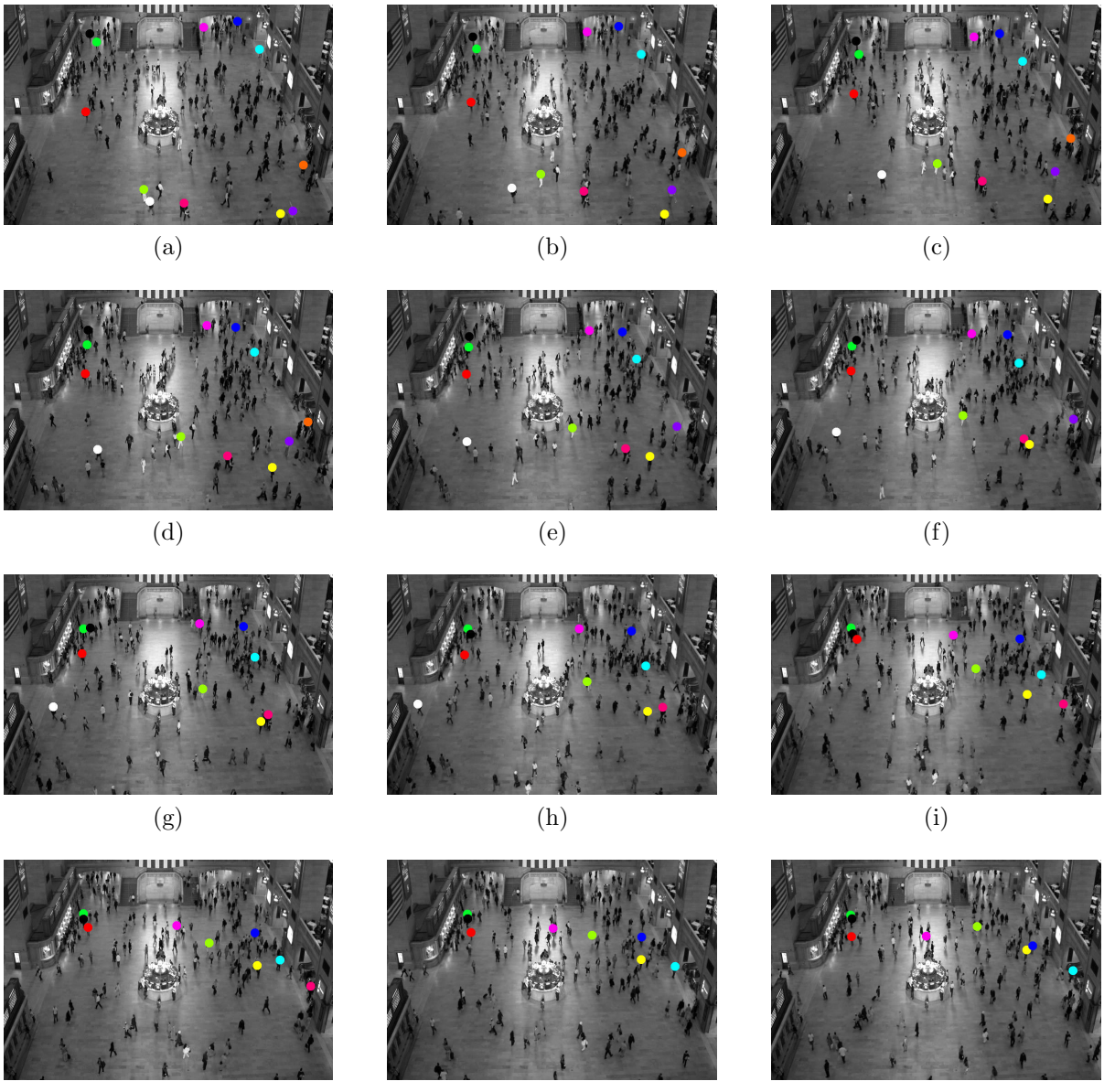
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.6: Shows frames from the synthesized campus video demonstrating the type of motion present. The frames shown are from the highest scoring synthetic video (15 agents operating with loose goals and no path diversification). Color labels added to some agents to assist viewing.



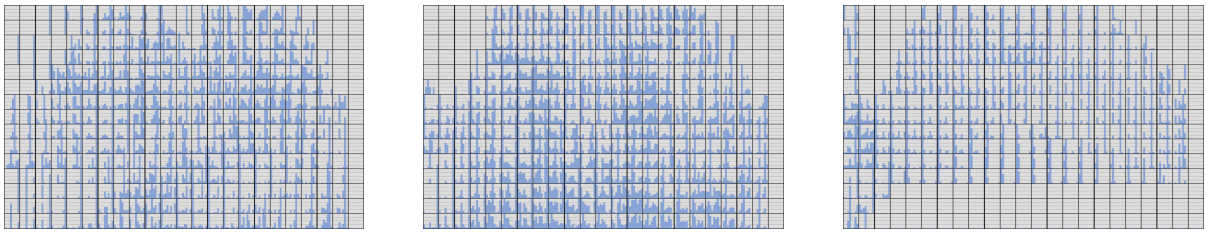
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.7: Shows frames from the Grand Central video (original) demonstrating the type of motion present. It should be noted the Grand Central video contains dense crowds and is shot from slight angle. Color labels added to some agents to assist viewing.



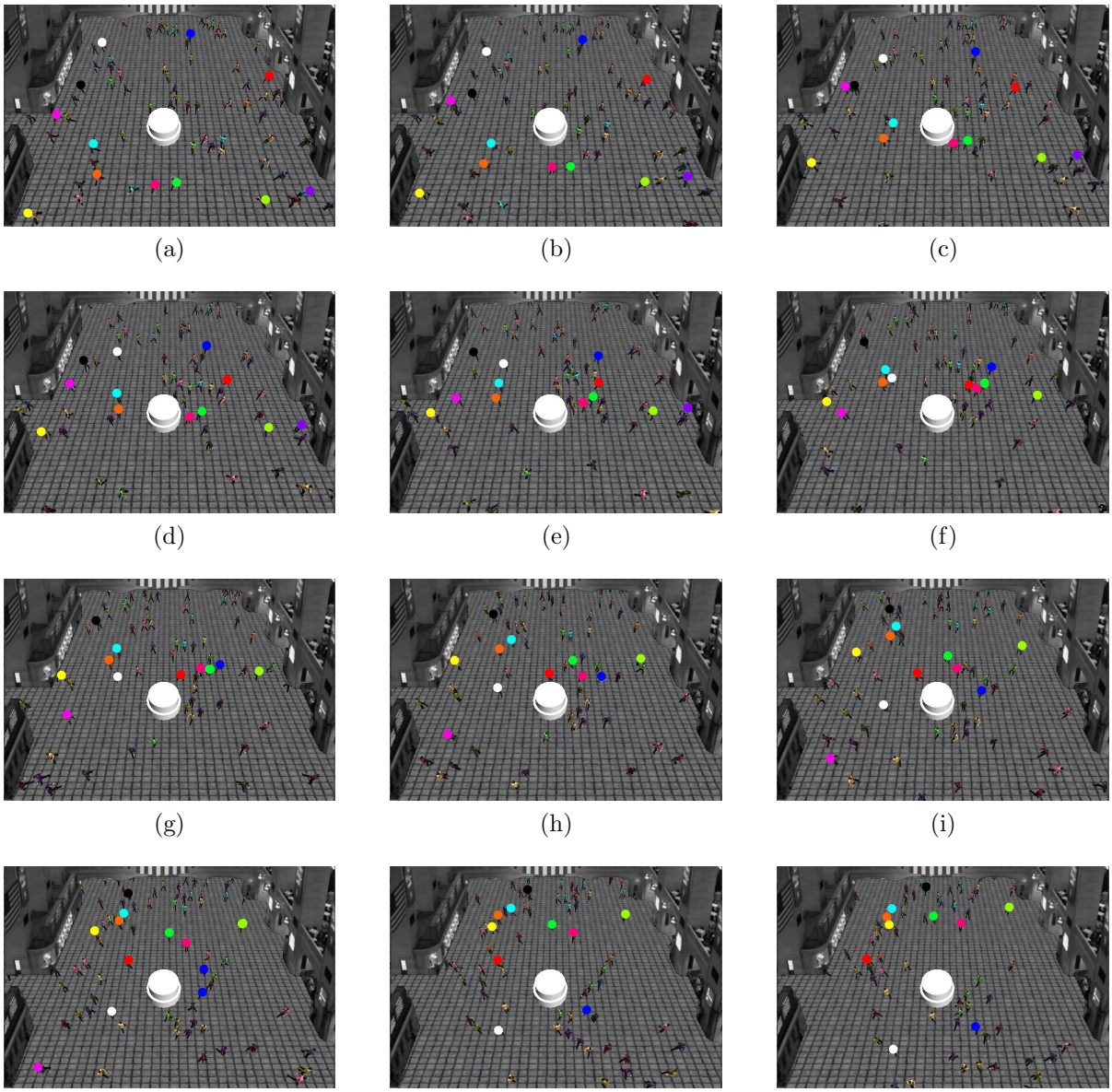
(a) (b) (c)

Figure 5.8: Grand central video histograms. (a) Histograms of Motion from the real video. (b) Histograms of motion from the best (15 agents, loose goals, no diversification) synthetic crowd. (c) Histograms of motion from the worst (05 agents, random paths) synthetic crowd.

This dataset has mixed results. The worst performing scenario being 50 agents operating with tight goals and no diversification. The best performing scenario being 100 agents operating with loose goals and diversification. The worst and best scores being 0.6091 and 0.4082 respectively. The best score obtained with this dataset is what should be expected from human trials as the agents flow from point to point more naturally while still following the dominant path logic observed in the input video. This is promising. However, the expected worst case would be random paths. This did not occur but this input crowd is a naturally random crowd. Although dominant paths can be extracted, the input grand central station crowd is essentially an open floor with pedestrians moving in many directions. The tight goals and no diversification used in the worst tested scenario seems to be more restrictive than giving agents random goals, thus resulting in a worse score.

5.2.3 UCF Crowd Video

Twelve synthetic crowds are created from the UCF Crowd video. The camera here is much closer (see Figure 5.10) to the pedestrians than the other two. The metric performs poorly on this dataset. Results (Table 5.4) are mixed, and it generally favors random motions. However, the original video has very unpredictable motions for pedestrians. The



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.9: Shows frames from the synthetic Grand Central video demonstrating the type of motion present. The frames shown are from the highest scoring synthetic video (100 agents operating with loose goals and diversification). Color labels added to some agents to assist viewing.

(a) (b) (c)

(d) (e) (f)

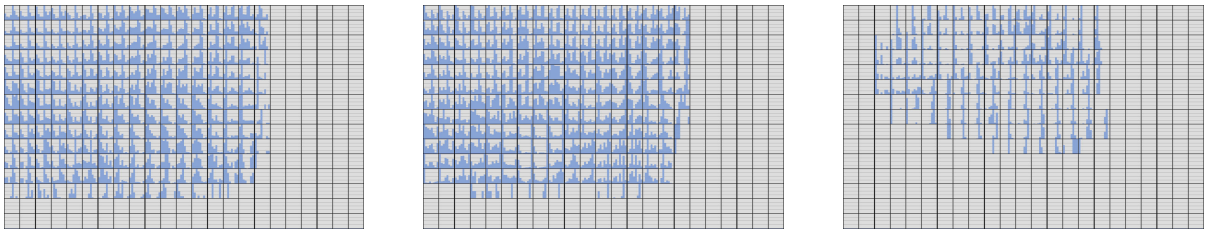
(g) (h) (i)

(j) (k) (l)

Figure 5.10: Shows some frames of the UCF Crowd Dataset video (original) to demonstrate the types of motion present. It should be noted that this video contains dense crowds captures close and from a slight angle. Color labels added to some agents to assist viewing.

Synthetic Crowd Characteristics	Score
10 agents, random paths	0.2573
20 agents, random paths	0.2428
30 agents, random paths	0.2423
10 agents, tight goals, no diversification	0.3774
20 agents, tight goals, no diversification	0.2858
30 agents, tight goals, no diversification	0.4390
10 agents, loose goals, no diversification	0.3508
20 agents, loose goals, no diversification	0.3328
30 agents, loose goals, no diversification	0.2436
10 agents, loose goals, diversification	0.3254
20 agents, loose goals, diversification	0.2949
30 agents, loose goals, diversification	0.3026

Table 5.4: Twelve synthetic crowds of the UCF Crowd Dataset and their scores based on Histograms of Motion.



(a) (b) (c)

Figure 5.11: UCF video histograms. (a) Histograms of Motion from the real video. (b) Histograms of motion from the best (30 agents, random paths) synthetic crowd. (c) Histograms of motion from the worst (30 agents, tight goals, no diversification) synthetic crowd.

best performing synthetic scenario is 30 agents operating on random paths, resulting in a score of 0.2423. The worst performing scenario was 30 agents operating with tight goals and no path diversification. The difference between the best and worst score makes sense as the input video has a very random naturally occurring crowd while the tight goals and no diversification simulation is very restricted. However, in this scenario random paths performed the best. It should be noted that the difference in score between the best random scenario and the best processed scenario is only 0.0013.

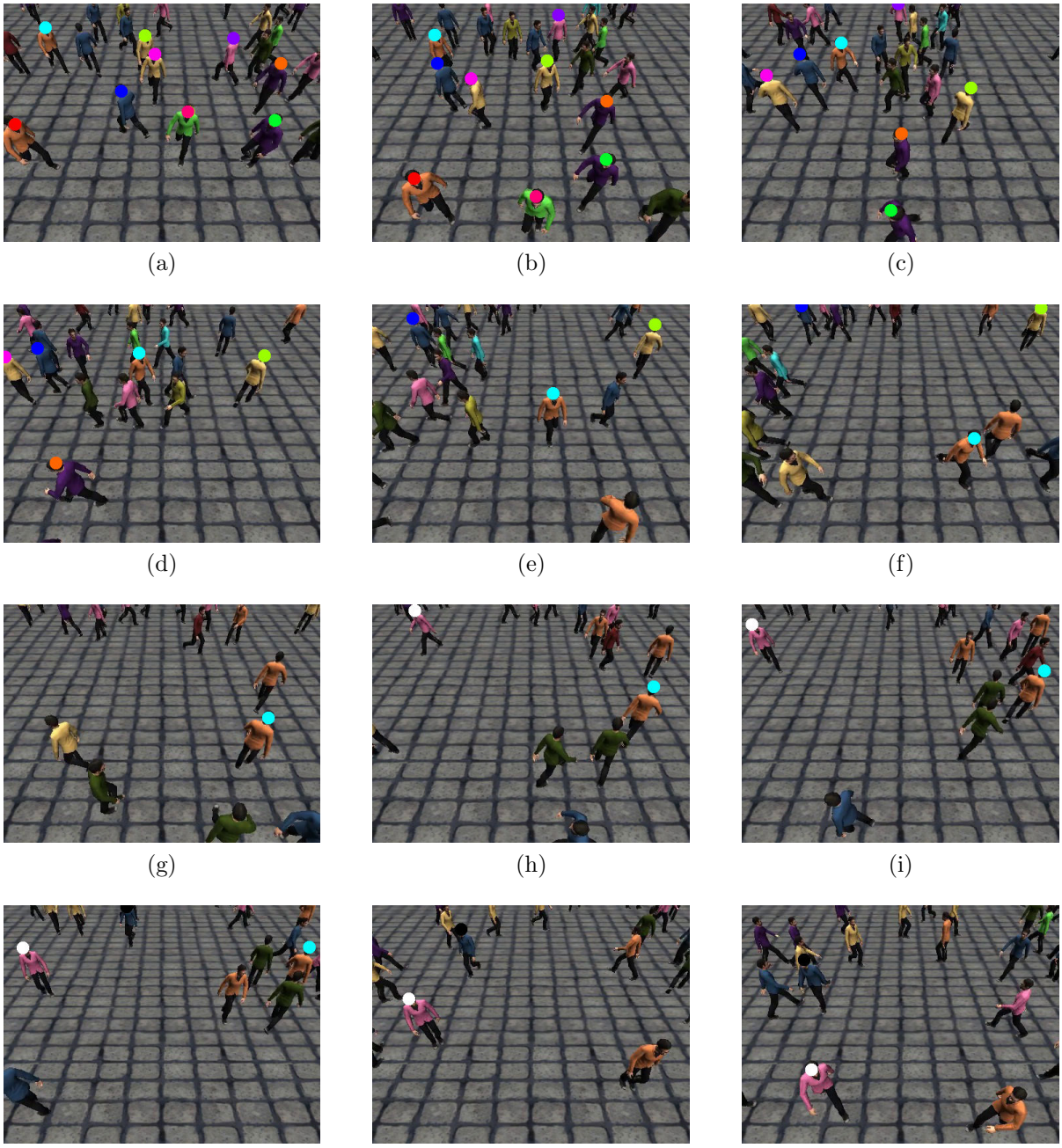


Figure 5.12: Shows frames from the synthetic UCF walking video demonstrating the type of motion present. The frames shown are from the highest scoring synthetic video (30 agents operating on random paths). Color labels added to some agents to assist viewing.

5.2.4 Histogram of Motion Score Discussion

Using the histogram of motion has mixed results as a metric for determining which crowd is the best synthesized version of a real crowd. A solid metric is needed to transform this framework from its current linear form into an iterative self-tuning framework. More work is needed in developing a metric that can satisfy this need and contribute to the usefulness of this system. From preliminary results presented here, it seems that our metric provides a mechanism to order videos according to their similarity to the exemplar videos.

There are some limitations to the metric. Firstly, temporal information is disregarded in the global motion vector collection process which is used for comparison between scenes. This could present a problem in time sensitive scenarios such as a cross walk at a red light. The resulting crowds might appear similar using the score, however the resulting crowds might disobey rules of the environement. The metric examines distributions of motion between similar areas of the scene, this distribution of motion could be reproduced with a different series of actions. Although the disribution of motion could be exactly the same, this does not mean that what is observed is exactly the same. Similarly, the metric is capable of producing a score of 0, which is misleading in that a score of 0 leads the user to believe an exact reproduction has occured. This however is not the case. A score of 0 could be generated with a very different series of motions, it just means that each local observation matched what was observed between videos.

With a stronger metric, self-tuning of the framework's various parameters could be performed. In this way the system would be able to iteratively perform the synthetic video production in hopes of outputting a synthetic result that resembles the input video strongly. The system has many parameters relating to the motion vector extraction method, motion vector clustering, path interpolation, animation state, crowd simulation, and crowd comparison. Automating the process of fine-tuning these parameters is highly desirable.

Chapter 6

Conclusions

We presented a framework for crowd simulation which accepts unstructured videos of crowds and through crowd analysis produces a synthetic version of that crowd. The output of the crowd analysis stage is used both to perform crowd synthesis and crowd comparison between real and synthetic crowds. We generate histograms of motion from crowds (real and synthetic) for the purpose of comparison. We perform similarity measures on real and our synthetic crowd videos to generate scores based on the Bhattacharyya distance which tells us how well the original crowd's motions are being represented. Using these scores we have shown that by making manual adjustments to the system we can minimize the similarity score and produce more similar crowd reproductions.

6.1 Future Work

The proposed framework operates under a very modular design making it easily adaptable to future uses. The pipeline itself starts and ends with a crowd video. The original plan was to make the system self-tuning. By this we mean the system would be able to accept a video of a crowd as input as it does now and reach a point where it has an output video. With the proposed self-tuning method, the system would then evaluate

the synthetic crowd in comparison to the original crowd and make its own adjustments to create better crowd simulations. The system would iteratively evolve itself to reach a more optimal solution. In the systems current state, these adjustments and manipulations to the system are performed manually. The core idea with the self-tuning approach is to have minimal input from the user for the purpose of reproducing a crowd.

Ground detection and automated camera positioning is a feature that did not make it into the current system. The user currently is responsible for ensuring this is correct. The implementation of automated camera positioning over the synthetic crowd is a requirement for the system to reach a stage of self-tuning.

The current set of animations is very basic. Agents are able to transition between walk and run states, with turns either right or left. They are also able to perform an idle animation whereby they shift their weight and perform subtle motions while remaining in a single position. Having a more intricate set of animations is desirable for our system. Incorporating some form of interaction between agents (talk gestures, shake hands, etc.) would help the simulation feel more realistic. Similarly having agents interact with the environment (sit on bench, stop and look at something in the scene, react to emergency)would contribute to the feeling of a reality and further improve the quality of our simulation. Even having the agents perform various animations with their upper body (talk on cell phone, wave, check watch) while navigating the scene would make the agents feel less robotic. We also need to be able to handle highly dense synthetic crowds.

6.1.1 Use Cases

The framework can be adapted to a number of use cases. The framework was designed in a modular manner resulting in a pipeline of stages. Due to its design, the framework is very adaptable and can easily be customize to various applications. The addition or subtraction of a module can be performed at a very high level and modules have the potential to be shared. In this way the system has the potential to be fine-tuned to

various scenarios while remaining minimal in execution. Although the original purpose was to save effort on the side of animators, we could see use cases in design planning, emergency planning, virtual reality, and crowd directing to name a few.

The proposed system could be adapted to help design spaces and ensure appropriate emergency responses. The system can accept a video of a space to observe and simulate how that space is being used by crowds. At this point changes can be made in the simulation to test environmental changes and see how the crowd might react. A good example of this might be to observe a crowd in a train station and synthesis this crowd. Observing the crowd and taking note of congestion, the designer would be able to make environmental adjustments such as the inclusion of an extra exit or two, to see if it helps with the crowd flow. Similarly tests could be performed with the exclusion of an exit. The idea being that the results of these simulations would help them design future spaces or make adjustments to current spaces that are sub-optimal. Testing the inclusion or exclusion of exits would require minor adjustments to the agent and pathing logic from it's current state.

The application of synthetic crowds generated from real world scenarios has some interesting applications in virtual reality. Immersing users in virtual crowds would be an interesting adaptation of the system. Seeing what observations the users makes of the crowd from the perspective of the crowd versus the perspective of the all-seeing (overhead).

Lastly, an interesting application of the system would be using a synthesized crowd as a data-driven simulation to manipulate a real-world environment. Observing how people are moving and interacting in an environment and then making changes to see if people become any more (or less) predictable. This might also have applications in emergency simulation and response.

Chapter 7

Appendices

7.1 Appendix A - Configuration File

The configuration file allows the user to configure any parameter in the system (both the preprocessing and simulation parameters can be set here).

video parameters relating to the input video

- **video** the video file location

videopreprocessing parameters relating to the preprocessing of the video

- **optical_flow** which type of flow vectors to generate (sift, dense, sparse, random)
- **start_frame** frame to start processing video from
- **end_frame** frame to finish processing video on
- **output_grid**
- **min_vector_length** minimum length to accept vectors (discarded otherwise)
- **every_n_frames** frame skipping parameter
- **sparse_winx** sparse: window width
- **sparse_winy** sparse: window height
- **sparse_maxLevel** sparse: 0-based maximal pyramid level number

sparse_minEigThres sparse: threshold to filter out flow values and increase performance

dense_pyr_scale dense: image scale to build pyramids

dense_level dense: number of pyramid layers (including original image)

dense_winsize dense: the size of window used for averaging

dense_iterations dense: iterations performed per pyramid level

dense_poly_n dense: size of pixel neighborhood

dense_poly_sigma dense: standard deviation of the Gaussian used in optical flow

sift_nfeatures sift: number of feature to retain

sift_nOctaveLayers sift: number of layers in each octave

sift_contrastThreshold sift: the threshold used to filter out weak points

sift_edgeThreshold sift: filter out edge like features

sift_sigma sift: sigma for the Gaussian applied to initial image

sift_threshold sift: threshold for rejection

sift_match_perc sift: percentage of matches to keep

navgrid navigation grid parameters

output Boolean, output grid if 1

bin_size bin size for processing

simulation simulation parameters

cam_pos_x camera x position

cam_pos_y camera y position

cam_pos_z camera z position

cam_rot_x camera x rotation

cam_rot_y camera y rotation

cam_rot_z camera z rotation

cam_scl_x camera x scale

cam_scl_y camera y scale

cam_scl_z camera z scale

7.2 Appendix B - Self-Tuning Spectral Clustering Algorithm

Algorithm: Given a set of points $S = s_1, ..., s_n$ in R^l that we want to cluster

1. Compute the local scale σ_i for each point $s_i \in S$ using $\sigma_i = d(s_i, s_K)$ where s_K is the Kth neighbor of s_i.

2. Form the locally scaled affinity matrix $\hat{A} \in R^{n \times n}$ where $\hat{A}_{ij}$ is defined according to $\hat{A}_{ij} = exp(\frac{-d^2(s_i, s_j)}{\sigma_i \sigma_j})$ for $i \neq j$ and $\hat{A}_{ii} = 0$.

3. Define D to be a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} \hat{A}_{ij}$ and construct the normalized affinity matrix $L = D^{-1/2} \hat{A} D^{-1/2}$.

4. Find $x_1, ..., x_C$ the C largest eigenvectors of L and form the matrix $X = [x_1, ..., x_C] \in R^{n \times C}$, where C is the largest possible group number.

5. Recover the rotation R which best aligns X's columns with the canonical coordinate system using the incremental gradient descent scheme.

6. Grade the cost of the alignment for each group number, up to C, according to $J = \sum_{i=1}^{n} \sum_{j=1}^{C} \frac{Z_{ij}^2}{M_i^2}$.

7. Set the final group number C_{best} to be the largest group number with minimal alignment cost.

8. Take the alignment result Z of the top C_{best} eigenvectors and assign the original point s_i to cluster c if and only if $max(Z_{ij}^2) = Z_{ic}^2$.

9. If highly noisy data, use the previous step result to initialize k-means, or EM, clustering on the rows of Z.

Taken from [56].

7.3 Appendix C - Globally Dominant Path Stitching Algorithm

While scanning, for each local motion flow,

1. Determine the neighbor cells, Ns. (See Figure 3.12)
2. In each N, search for the motion flows that are in the same orientation group
3. Choose the closest one in the neighborhood and connect with the current flow.
4. If, there are not motion flows with the same orientation group in the neighbor cells and next neighbor cells, choose the motion flow that is the closest

Taken from [36].

Bibliography

[1] Maria Andersson, Joakim Rydell, Louis St-Laurent, Donald Prévost, and Fredrik Gustafsson. Crowd analysis with target tracking, k-means clustering and hidden markov models. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 1903–1910. IEEE, 2012.

[2] Ernesto L Andrade, Scott Blunsden, and Robert B Fisher. Modelling crowd scenes for event detection. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 175–178. IEEE, 2006.

[3] Nikolai WF Bode, Jolyon J Faria, Daniel W Franks, Jens Krause, and A Jamie Wood. How perceived threat increases synchronization in collectively moving animal groups. *Proceedings of the Royal Society B: Biological Sciences*, 277:3065–3070, 2010.

[4] Matthias Butenuth, Florian Burkert, Florian Schmidt, Stefan Hinz, Dirk Hartmann, Angelika Kneidl, André Borrmann, and Beril Sirmaçek. Integrating pedestrian simulation, tracking and event detection for crowd analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 150–157. IEEE, 2011.

[5] Stephen Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242. Eurographics Association, 2004.

[6] Hannah M Dee and Alice Caplier. Crowd behaviour analysis using histograms of motion direction. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 1545–1548. IEEE, 2010.

[7] Günther Eibl and N Brandle. Evaluation of clustering methods for finding dominant optical flow fields in crowded scenes. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[8] Ran Eshel and Yael Moses. Tracking in a dense crowd using multiple cameras. *International journal of computer vision*, 88(1):129–143, 2010.

[9] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*, pages 363–370. Springer, 2003.

[10] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[11] Matthew Flagg and James M Rehg. Video-based crowd synthesis. *Visualization and Computer Graphics, IEEE Transactions on*, 19(11):1935–1947, 2013.

[12] Geometric algorithms for modeling, motion, and animation, August 2014.

[13] Carolina Garate, Piotr Bilinsky, and François Bremond. Crowd event recognition using hog tracker. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1–6. IEEE, 2009.

[14] Crowd simulation for maya — golaem crowd, August 2014.

[15] Stephen J Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Transactions on Graphics (TOG)*, 31(6):190, 2012.

[16] Min Hu, Saad Ali, and Mubarak Shah. Learning motion patterns in crowded scenes using motion flow field. In *ICPR*, pages 1–5, 2008.

[17] Nacim Ihaddadene and Chabane Djeraba. Real-time crowd motion analysis. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[18] Ioannis Karamouzas and Mark Overmars. Simulating and evaluating the local behavior of small pedestrian groups. *Visualization and Computer Graphics, IEEE Transactions on*, 18(3):394–406, 2012.

[19] Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee. Tiling motion patches. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, pages 117–126. Eurographics Association, 2012.

[20] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM transactions on graphics (TOG)*, 21:473–482, 2002.

[21] Barbara Krausz and Christian Bauckhage. Analyzing pedestrian behavior in crowds for automatic detection of congestions. In *Computer vision workshops (ICCV workshops), 2011 IEEE international conference on*, pages 144–149. IEEE, 2011.

[22] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–118. Eurographics Association, 2007.

[23] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: A data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 109–118, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[24] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion patches: building blocks for virtual environments annotated with motion data. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 898–906. ACM, 2006.

[25] Alon Lerner, Eitan Fitusi, Yiorgos Chrysanthou, and Daniel Cohen-Or. Fitting behaviors to pedestrian simulations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 199–208. ACM, 2009.

[26] Yi Li, Marc Christie, Orianne Siret, Richard Kulpa, and Julien Pettré. Cloning crowd motions. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 201–210. Eurographics Association, 2012.

[27] Honghong Liao, Jinhai Xiang, Weiping Sun, Qing Feng, and Jianghua Dai. An abnormal event recognition in crowd scene. In *Image and Graphics (ICIG), 2011 Sixth International Conference on*, pages 731–736. IEEE, 2011.

[28] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[29] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[30] Massive software - simulating life, August 2014.

[31] 3d animation software, computer animation software — maya — autodesk, August 2014.

[32] Ramin Mehran, Alexis Oyama, and Mubarak Shah. Abnormal crowd behavior detection using social force model. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 935–942. IEEE, 2009.

[33] Miarmy - basefount miarmy maya crowd simulation tools, August 2014.

[34] Brian E Moore, Saad Ali, Ramin Mehran, and Mubarak Shah. Visual crowd surveillance through a hydrodynamics lens. *Communications of the ACM*, 54:64–73, 2011.

[35] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. In *ACM Transactions on Graphics (TOG)*, volume 29, page 123. ACM, 2010.

[36] Ovgu Ozturk, Toshihiko Yamasaki, and Kiyoharu Aizawa. Detecting dominant motion flows in unstructured/structured crowd scenes. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3533–3536. IEEE, 2010.

[37] Sachin Patil, Jur Van Den Berg, Sean Curtis, Ming C Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *Visualization and Computer Graphics, IEEE Transactions on*, 17:244–254, 2011.

[38] Nuria Pelechano, Catherine Stocker, Jan Allbeck, and Norman Badler. Being a part of the crowd: towards validating vr crowds using presence. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 136–142. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[39] Stefano Pellegrini, Jürgen Gall, Leonid Sigal, and Luc Van Gool. Destination flow for crowd simulation. In *Computer Vision–ECCV 2012. Workshops and Demonstrations*, pages 162–171. Springer, 2012.

[40] Wei-Ya Ren, Guo-Hui Li, Jun Chen, and Hao-Zhe Liang. Abnormal crowd behavior detection using behavior entropy model. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2012 International Conference on*, pages 212–221. IEEE, 2012.

[41] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21:25–34, 1987.

[42] Mikel Rodriguez, Saad Ali, and Takeo Kanade. Tracking in unstructured crowded scenes. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1389–1396. IEEE, 2009.

[43] Mikel Rodriguez, Josef Sivic, Ivan Laptev, and J-Y Audibert. Data-driven crowd analysis in videos. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1235–1242. IEEE, 2011.

[44] Shobhit Saxena, François Brémond, Monnique Thonnat, and Ruihua Ma. Crowd behavior recognition for video surveillance. In *Advanced Concepts for Intelligent Vision Systems*, pages 970–981. Springer, 2008.

[45] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28. ACM, 2005.

[46] Hubert PH Shum, Taku Komura, Masashi Shiraishi, and Shuntaro Yamazaki. Interaction patches for multi-character animation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 114. ACM, 2008.

[47] David JT Sumpter. The principles of collective animal behaviour. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 361:5–22, 2006.

[48] Libo Sun and Wenhu Qin. A data-driven approach for simulating pedestrian collision avoidance in crossroads. In *Digital Media and Digital Content Management (DMDCM), 2011 Workshop on*, pages 83–85. IEEE, 2011.

[49] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[50] Ian M Thornton and Quoc C Vuong. Incidental processing of biological motion. *Current Biology*, 14:1084–1089, 2004.

[51] Unity - game engine, August 2014.

[52] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.

[53] Xin Wang and Shouqian Sun. Data-driven macroscopic crowd animation synthesis method using velocity fields. In *Computational Intelligence and Design, 2008. ISCID'08. International Symposium on*, volume 2, pages 157–160. IEEE, 2008.

[54] Guogang Xiong, Xinyu Wu, Yen-Lun Chen, and Yongsheng Ou. Abnormal crowd behavior detection based on the energy model. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 495–500. IEEE, 2011.

[55] Barbara Yersin, Jonathan Maïm, Julien Pettré, and Daniel Thalmann. Crowd patches: populating large-scale virtual environments for real-time applications. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 207–214. ACM, 2009.

[56] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *NIPS*, volume 17, page 16, 2004.

[57] Beibei Zhan, DorothyN. Monekosso, Paolo Remagnino, SergioA. Velastin, and Li-Qun Xu. Crowd analysis: a survey. *Machine Vision and Applications*, 19(5-6):345–357, 2008.